UNIVERSITY OF CALIFORNIA, SAN DIEGO

On the Fixed-Point Analysis and Architecture Design of FFT Algorithms

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Electrical Circuits and Systems)

by

Wei-Hsin Chang

2007

The dissertation of Wei-Hsin Chang is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2007

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I have depended on the help and support of many people, to whom I owe far more gratitude than I can possibly express here.

I would like to express my deepest gratitude to my advisor Prof. Truong Q. Nguyen for his inspiring suggestions and stimulating encouragements in all the time of research, which directly enable the completion of this dissertation.

Besides, I would like to thank UCSD Center for Wireless Communication (CWC), UC matching fund, National Semiconductor Corporation (NSC) and Texas Instruments Inc. (TI) for their financial support during the past four years. Also, I appreciate all members of Video Processing Lab for plenty of substantial discussion and suggestionsfrom them. Besides, I would like to thank Dr. Toshio Nagata, Edwin Park, Ziad Asghar, and other TIers for their for their constructive feedbacks regarding the cooperation project between TI and UCSD.

Last but not least, I would lik to thank my parents and my brother who all give me their sincere support and assistance to go through this important stage of my life.

VITA

August, 7th, 1977	Born, Taichung City, Taiwan, R.O.C
1999	B.S, Department of Electrical Engineering, National Tsing-Hua University
1999–2001	Teaching Assistant, Department of Electronics Engineering, National Chiao-Tung University
2001	M.S, Electronics Engineering, National Chiao-Tung University
2003-2007	Research Assistant, University of California, San Diego
2007	Ph. D, University of California, San Diego

PUBLICATIONS

Wei-Hsin Chang and Truong Nguyen, “On the Fixed-Point Accuracy Analysis of FFT Algorithms“, submitted to Trans. on Signal Processing, May 2007.

Wei-Hsin Chang and Truong Nguyen, “Integer FFT with Optimized Coefficient Sets“, To be appeared on Acoustics, Speech and Signal Processing, 2007. ICASSP 2007 Proceedings. 2007 IEEE International Conference on.

Wei-Hsin Chang and Truong Nguyen, “An OFDM-specified lossless FFT architecture” Circuits and Systems I: Regular Papers, IEEE Transactions on, Volume 53, Issue 6, June 2006 Page(s):1235 - 1243.

Wei-Hsin Chang and Truong Nguyen, “Architecture and Performance Analysis of Lossless FFT in OFDM Systems“, Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on, Volume 3, 2006 Page(s):III - III.

Keng-Khai Ong, Wei-Hsin Chang, Yi-Chen Tseng, Yew-San Lee and Chen-Yi Lee, “A high throughput low cost context-based adaptive arithmetic codec for multiple standards“, Image Processing. 2002. Proceedings. 2002 International Conference on, Volume 1, 22-25 Sept. 2002 Page(s):I-872 - I-875 vol.1.

Keng-Khai Ong, Wei-Hsin Chang, Yi-Chen Tseng, Yew-San Lee and Chen-Yi Lee, “A high throughput context-based adaptive arithmetic codec for JPEG2000“, Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on Volume 4, 26-29 May 2002 Page(s):IV-133 - IV-136 vol.4.

Wei-Hsin Chang, Yew-San Lee, Wen-Shiaw Peng and Chen-Yi Lee, “A line-based, memory efficient and programmable architecture for 2D DWT using lifting scheme“, Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on Volume 4, 6-9 May 2001 Page(s):330 - 333 vol. 4.

Wei-Hsin Chang, Shuenn-Der Tzeng and Chen-Yi Lee, “A novel subcircuit extraction algorithm by recursive identification scheme“, Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on Volume 5, 6-9 May 2001 Page(s):491 - 494 vol. 5.

Yew-San Lee, Cheng-Mou Yu, Wei-Hsin Chang and Chen-Yi-Lee, “HVLC: error correctable hybrid variable length code for image coding in wireless transmission“, Acoustics, Speech, and Signal Processing, 2000. ICASSP ’00. Proceedings. 2000 IEEE International Conference on Volume 6, 5-9 June 2000 Page(s):2103 - 2106 vol.4.

Yew-San Lee, Wei-Hsin Chang, Hsin-Han Ho and Chen-Yi Lee, “Construction of error resilient synchronization codeword for variable-length code in image transmission“, Image Processing, 2000. Proceedings. 2000 International Conference on Volume 3, 10-13 Sept. 2000 Page(s):360 - 363 vol.3.

FIELDS OF STUDY

Major Field: Engineering

Studies in Electrical and Computer Engineering.
Professor Truong Q. Nguyen

ABSTRACT OF THE DISSERTATION

On the Fixed-Point Analysis and Architecture Design of FFT Algorithms

by

Wei-Hsin Chang
Doctor of Philosophy in Electrical Engineering (Electrical Circuits and Systems)

University of California San Diego, 2007

Professor Truong Q. Nguyen, Chair

In this thesis, first we investigate the principle of finding the optimized coefficient set of integer FFT (IntFFT). IntFFT has been regarded as an approximation of original FFT computation since it utilizes lifting scheme (LS) and decomposes the complex multiplication of twiddle factor into three lifting steps. Based on the observation of the quantization loss model of lifting operations, we can select an optimized coefficient set and achieve better signal-to-quantization-noise ratio (SQNR) and reduce size of coefficient table. Secondly, we analyze the fixed-point effect of arithmetic quantization errors for different fast Fourier transform (FFT) algorithms. A general analytic expression is derived to quantitatively compare the overall quantization loss. An operational optimization procedure is also proposed to find the optimal memory setting for short-length FFT architecture. Last, a parallel very-large-scale integration (VLSI) architecture based on mixed-radix IntFFT for the upcoming multiband OFDM (MB-OFDM) system is proposed. The periodicity property of lifting coefficients and the concurrent relationship of non-trivial multiplications are both utilized to reduce the hardware cost of complex multipliers.

1

Introduction

This chapter gives a brief introduction of discrete Fourier transform (DFT), the history of its developement, related applications, and the recent works on efficient FFT algorithms.

1.1 Motivation

DFT-based signal processing plays a significant role in today's digital signal processing (DSP) applications. It has been applied in a wide range of fields such as noise reduction, global motion estimation [1], asymmetrical digital subscriber line (ADSL), digital audio broadcasting (DAB), digital video broadcasting (DVB) [2–5], orthogonal frequency division multiplexing (OFDM) systems. The rapidly increasing demand of OFDM-based applications for high-speed wireless data communication, including wireless LAN [6], and MB-OFDM [7–9], makes processing speed an important major consideration in FFT architecture design. Besides, since split-radix FFT and other higher-radix FFT algorithms are proved to be computationally efficient than conventional Cooley-Tukey algorithm [10] and are widely used for hardware implementation, as such the study of finite precision effect of all FFT algorithms is of increasing significance.

1.2 History of FFT

The main research issues of FFT algorithms include three aspects: the efficient decomposition of DFT to reduce the multiplicative complexity by extracting as many trivial complex multiplications as possible, the fixed-point effect of FFT algorithms due to the arithmetic quantization errors, and the high-speed and/or low-cost VLSI architecture of FFT algorithms.

The original computation of DFT with N-sample input requires N^2 complex multiplications. Cooley and Tukey [10] first introduced the concept of FFT to demonstrate a significant computational reduction from $O(N^2)$ to $O(N \log N)$ by making efficient use of symmetry and periodicity properties of the twiddle factors. Since the pioneering work of Cooley-Tukey algorithm [10], several algorithms have been developed to further reduce the computational complexity, including radix-4 [11], radix-2^2 [12] and split-radix [13]. In common, these fast algorithms recursively divide the FFT computation into odd- and even-half parts recursively and then extract as many common twiddle factors as possible. Other works have also been done for high-radix FFT algorithms such as radix-8 FFT algorithm [14–16]. Generally speaking, the number of required real additions and multiplications is the general measurement used to compare the efficiency of different FFT algorithms.

The second issue concerns the finite length limitation of digital computers. In practice, fixed-point arithmetic is used to implement FFT algorithms in hardware because it is not possible to keep infinite resolution of coefficients and operations. All coefficients and input signals have to be represented by finite number of bits in binary format depending on the tradeoff between the hardware cost (memory usage) and the accuracy of output signals. In general, each multiplication may introduce an error due to rounding operation or truncation, which is regarded to arithmetic quantization error. Besides, all the twiddle factors are represented with limited number of bits and the loss due to the inexact coefficients is called coefficient quantization error. The theoretical performance evaluation has been given in previous works. Several existent works have analyzed the effect of fixed-point arithmetic for radix-2 FFT [17–20]. James [21] derived the fixed-point mean square error (MSE) analysis of quantization loss for mixed-radix FFT algorithms with conventional complex multipliers. Perlow and Denk [22] proposed an error propagation model to estimate the performance for radix-2 FFT architecture. Instead of conventional implementation of complex multiplier, Chandra [23] studied the same issue for radix-2 decimation-in-time (DIT) FFT in the logarithmetic numbering system. Park and Cho [24] used a propagation model to address the error analysis for coordinate rotation digital computer (CORDIC) implementations. Chang [25] also gave an experimental comparison of FFT computation with different implementations of complex multiplier.

In the VLSI era, not only the multiplicative complexity but also the hardware cost as well as the processing speed are important considerations. Numerous researches have been done for FFT architectures in the past decades. As for the implementation of FFT algorithms, most previous works address two design issues: the efficiency of complex multipliers and the memory strategy. The design parameters have to be adjusted according to different target applications. Designers focus on increasing the utilization rate of multipliers in the design of short-length FFT architecture. In [26], the authors proposed a coefficient reordering scheme with the commutator architecture to reduce the switching activity. Yeh and Jen [27] proposed a data rescheduling scheme by bit-reverse and bit-inverse indexing to reduce the number of required multipliers. Han et al. [28] achieve the same purpose by sharing the subexpressions between multiplicands to improve the utilization rate of complex multipliers. Conversely the memory access strategy becomes more important on designing a long-length FFT architecture. In [29] and [30], the authors increase the memory access efficiency by inserting two caches between the functional units and the main memory banks. Kuo et al. [31] also utilize the memory-cache architecture and expand it with the ability of variable-length FFT computations. The precision of output samples is another important issue for long-length FFT designs since the size of internal memory increases exponentially with the length of FFT. Choi et al. [32] proposed the coverage block floating point (CBFP) method to dynamically normalize the intermediate results after each stages. In [33], the authors also utilize the idea of CBFP with an on-the-fly normalization module so as to remove the usage of temporary buffers.

1.3 Recent Works on Efficient FFT Algorithms

Nowadays, different FFT algorithms and corresponding VLSI architecture designs are proposed to meet the requirement of multimedia applications and wireless applications. For the multiplicative complexity, a modified split-radix FFT is reported to have the fewest number of floating point operation (FLOP) counts [34]. The modified split-radix approach involves a recursive rescaling of the trigonometric constants in sub-transforms of the DFT decomposition, while the final FFT result is still the correct, relying on four mutually recursive stages.

The parallelism of FFT algorithms is also exploited in [35–38] to achieve higher throughput and lower computation latency. In [37], the author utilized the two-dimensional mapping of a 1-D DFT to compute long-length FFT in parallel. The complex multiplier between the two pipeline stages is eliminated to save hardware cost. A simplified address generation scheme of twiddle factors is also proposed to reduce the size of the table of twiddle factors. In [38], the authors take advantage of parallelism by using single instruction, multiple data (SIMD) instruction set.

2

Previous Work on FFT Algorithms

In this chapter, the definition of DFT is described. The applications of DFT in digital signal processing are also introduced. Several FFT algorithms including radix-2, radix-4 and split-radix algorithms are reviewed and compared in terms of their computational efficiency. The previous work of different implementation of complex multipliers including numerical decomposition, CORDIC operation and lifting scheme decomposition are also illustrated.

2.1 Definition of DFT

The computation of the DFT involves the multiplication of a twiddle factor matrix by a complex-valued input vector. Given an input sequence $x(n)$, the N-point DFT is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{nk}, \quad k = 0, 1, ..., N-1, \tag{2.1}$$

where n is the time index, k is the frequency index, and the twiddle factor W_N^{nk} is defined as

$$W_N^{nk} = exp(\frac{-2j\pi nk}{N}) = cos(\frac{2\pi nk}{N}) - j \cdot sin(\frac{2\pi nk}{N}). \tag{2.2}$$

Similarly, inverse discrete Fourier transform (IDFT) can be expressed as:

$$x(n) = \frac{1}{N} \cdot \sum_{n=0}^{N-1} X(n) \cdot W_N^{-nk}, \quad k = 0, 1, ..., N-1. \tag{2.3}$$

From (2.1) and (2.3), we can observe that N complex multiplications and $N-1$ complex additions are performed for one output sample. Therefore, N^2 complex multiplications and $N^2 - N$ complex additions are necessary to complete the whole DFT/IDFT computation with N-sample input vectors. The complexity of direct computation of DFT and IDFT is inefficient since it will increase exponentially according to N.

2.2 FFT Algorithms

In order to reduce the computational complexity of DFT, we can apply the periodicity and symmetrical properties of twiddle factors shown in (2.4) and (2.5) respectively to eliminate the redundant complex multiplications.

$$W_N^{k+N} = W_N^k \tag{2.4}$$

$$W_N^{k+\frac{N}{2}} = -W_N^k \tag{2.5}$$

2.2.1 Radix-2 FFT Algorithm

Divide-and-conquer paradigm refers to partitioning a larger problem set into several smaller ones and solve each smaller set recursively. The main procedure of divide-and-conquer paradigm include:

- Step 1: Partition the original problem set into two or more smaller subsets.
- Step 2: Solve the problem subsets by applying the same procedure recursively. A preset terminate condition is necessary to stop the recursion when the size of problem subsets is smaller enough to be solved directly.

- Step 3: Obtain the solution of original problem set by assembling the results of the smaller subsets.

The Cooley-Tukey radix-2 FFT algorithm is also a divide-and-conquer algorithm which recursively split the DFT computation into odd- and even-half parts. The original input vector, $x(n)$ is split into two $\frac{N}{2}$-length vectors, $x_e(n)$, and $x_o(n)$, as defined in (2.6) and (2.7).

$$x_e(n) = x(2n) \tag{2.6}$$

$$x_o(n) = x(2n+1),\ n = 0, 1, ..., \frac{N}{2} - 1 \tag{2.7}$$

Because $x_e(n)$ and $x_o(n)$ are obtained by decimating the time index of $x(n)$ by a factor of 2, the resulting algorithm is named DIT FFT algorithm. Otherwise, instead of the input vector, if the output vector is decimated in frequency index, the resulting algorithm is called decimation-in-frequency (DIF) FFT.
The radix-2 DIT FFT is derived by rewriting (2.1) as:

$$\begin{aligned} X(k) &= \sum_{m=0}^{\frac{N}{2}-1} x(2m) \cdot W_N^{2mk} + \sum_{m=0}^{\frac{N}{2}-1} x(2m+1) \cdot W_N^{(2m+1)k} \qquad (2.8) \\ &= \sum_{m=0}^{\frac{N}{2}-1} x_e(m) \cdot W_N^{2mk} + \sum_{m=0}^{\frac{N}{2}-1} x_o(m) \cdot W_N^{(2m+1)k},\ k = 0, 1, ..., N-1 \end{aligned}$$

Because $W_N^2 = W_{\frac{N}{2}}$, we can rewrite (2.8) as:

$$X(k) = \sum_{m=0}^{\frac{N}{2}-1} x_e(m) \cdot W_{\frac{N}{2}}^{mk} + W_N^k \sum_{m=0}^{\frac{N}{2}-1} x_o(m) \cdot W_{\frac{N}{2}}^{mk} \tag{2.9}$$

By periodicity property expressed in (2.4), it is clear that $W_{\frac{N}{2}}^{(k+\frac{N}{2})m} = W_{\frac{N}{2}}^{km}$. As a result, it is only necessary to calculate the summations for $k = 0, 1, ..., \frac{N}{2} - 1$. Hence, each summation in (2.9) can be interpreted as two DFTs of length-$\frac{N}{2}$, where the first summation involves the even-indexed sequence, $x_e(n)$ and the second summation involves the odd-indexed sequence, $x_o(n)$. Therefore, the original length-N

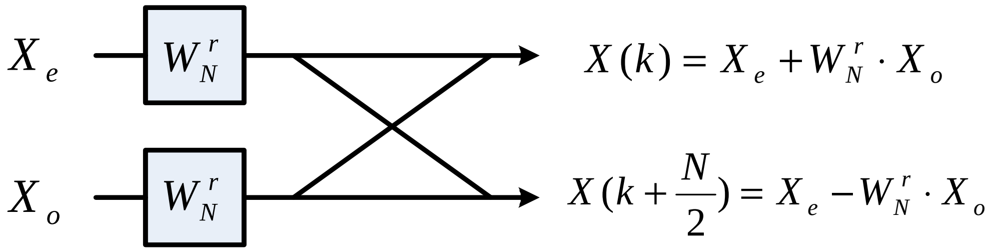

X_e
X_o
W_N^r
W_N^r
$X(k) = X_e + W_N^r \cdot X_o$
$X(k+\frac{N}{2}) = X_e - W_N^r \cdot X_o$

Figure 2.1: The butterfly stage of Cooley-Tukey DIT FFT.

DFT becomes two length-$\frac{N}{2}$ DFTs. Defining $X_e(k)$ and $X_o(k)$ as:

$$X_e(k) = \sum_{n=0}^{\frac{N}{2}-1} x_e(n) \cdot W_{\frac{N}{2}}^{nk}, \tag{2.10}$$

$$X_o(k) = \sum_{n=0}^{\frac{N}{2}-1} x_o(n) \cdot W_{\frac{N}{2}}^{nk}, \tag{2.11}$$

we can rewrite the first half terms of the original DFT as:

$$X(k) = X_e(k) + W_N^k \cdot X_o(k),\ k = 0, 1, ..., \frac{N}{2} - 1. \tag{2.12}$$

Similarly, by applying the fact that $W_N^{\frac{N}{2}+k} = -W_N^k$, the second half terms are given as:

$$\begin{aligned} X(k+\frac{N}{2}) &= \sum_{k=0}^{\frac{N}{2}-1} x_e(m) \cdot W^{(k+\frac{N}{2})m} + W_N^{k+\frac{N}{2}} \sum_{m=0}^{\frac{N}{2}-1} x_e(m) W_{\frac{N}{2}}^{(k+\frac{N}{2})m} \\ &= \sum_{m=0}^{\frac{N}{2}-1} x_e(m) \cdot W_{\frac{N}{2}}^{km} - W_N^r \sum_{m=0}^{\frac{N}{2}-1} x_o(m) \cdot W_{\frac{N}{2}}^{km} \\ &= X_e(k) - W_N^k \cdot X_o(k), k = 0, 1, ..., \frac{N}{2} - 1 \end{aligned} \tag{2.13}$$

The original problem set is solved by combining (2.12) and (2.13), which are commonly refered as Cooley-Tukey algorithm in the related literature and shown in Figure 2.1.

Similarly, the radix-2 DIF FFT algorithm is derived by decimating the output vector into an even-indexed set and an odd-indexed set. To define the two problem subsets, (2.1) is rewritten as:

$$\begin{aligned} X(k) &= \sum_{n=0}^{\frac{N}{2}-1} x(n)W_N^{nk} + \sum_{n=\frac{N}{2}}^{N-1} x(n)W_N^{nk} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(n)W_N^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x(n+\frac{N}{2})W_N^{(n+\frac{N}{2})k} \\ &= \sum_{n=0}^{\frac{N}{2}-1} [x(n)+x(n+\frac{N}{2})W_N^{r\frac{N}{2}}]W_N^{nk},\ k=0,1,...,N-1 \qquad (2.14) \end{aligned}$$

For k is even, we can define rewrite (2.14) as:

$$\begin{aligned} X(2m) &= \sum_{n=0}^{\frac{N}{2}-1} [x(n)+x(n+\frac{N}{2})W_N^{mN}]W_N^{2mn} \\ &= \sum_{n=0}^{\frac{N}{2}-1} [x(n)+x(n+\frac{N}{2})]W_{\frac{N}{2}}^{mn},\ m=0,1,...,\frac{N}{2} \qquad (2.15) \end{aligned}$$

Defining $X_e(m) = X(2m)$ and $x_e(n) = x(n) + x(n+\frac{N}{2})$, we can get the first-half problem subset as:

$$X_e(m) = \sum_{n=0}^{\frac{N}{2}-1} x_e(n)W_{\frac{N}{2}}^{mn},\ k=0,1,...,\frac{N}{2} \qquad (2.16)$$

Similarly, for k is odd, the odd-indexed output samples can be expressed as:

$$\begin{aligned} X(2m+1) &= \sum_{n=0}^{\frac{N}{2}-1} [x(n)+x(n+\frac{N}{2})W_N^{(2m+1)\frac{N}{2}}]W_N^{(2m+1)n} \\ &= \sum_{n=0}^{\frac{N}{2}-1} [x(n)-x(n+\frac{N}{2})]W_N^{n}W_{\frac{N}{2}}^{mn},\ m=0,1,...,\frac{N}{2} \qquad (2.17) \end{aligned}$$

Again, by defining $X_o(m) = X(2m)$ and $x_e(n) = (x(n) - x(n+\frac{N}{2}))W_N^n$, we can obtain the second-half problem subset as:

$$X_o(m) = \sum_{n=0}^{\frac{N}{2}-1} x_o(n)W_{\frac{N}{2}}^{mn},\ k=0,1,...,\frac{N}{2} \qquad (2.18)$$

As a result, by computing both $X_e(m)$ and $X_o(m)$, we can also obtain the result of the original DFT computation. Consider a 16-point FFT as example, the

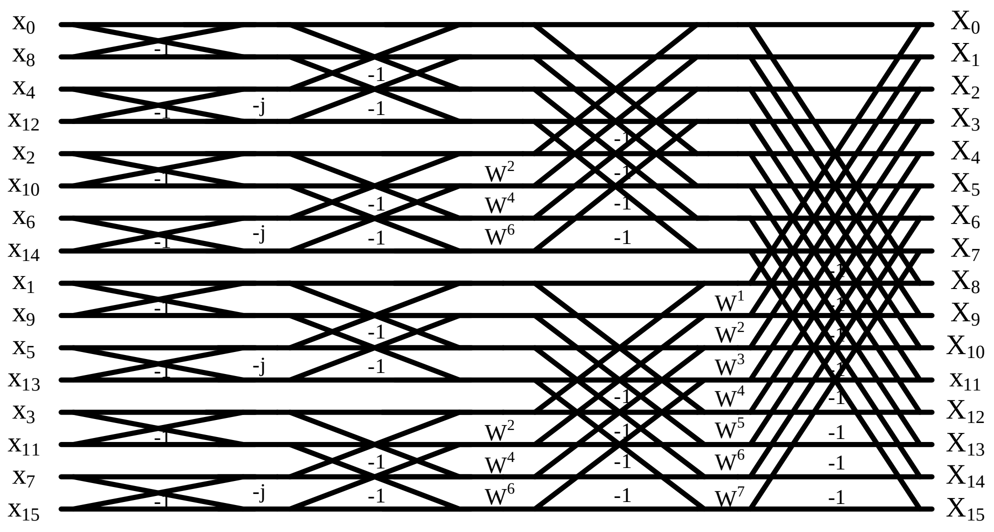


Figure 2.2: The signal flow diagram of 16-point radix-2 DIT algorithm.

signal flow diagrams of both radix-2 DIT and DIF FFT algorithms are shown in Figure 2.2 and Figure 2.3 respectively. As observed, the number of non-trivial complex multiplications of both schemes is the same, but the sequence of complex multiplications is in a reversed order.

2.2.2 Radix-4 FFT Algorithm

For DFT of a time series whose length N is power of 4, we can, of course, still apply radix-2 FFT algorithms. Nevertheless, in order to further reduce the multiplicative complexity, it is worthwhile to develop radix-4 FFT algorithms instead of simply using radix-2 FFT algorithms. The main idea of radix-4 DIT FFT is to divide the original input sequence into four smaller subsequences. By taking advantage of the identities that $W_N^{\frac{N}{4}} = -j$ and $W_N^4 = W_{\frac{N}{4}}$, (2.1) can be rewritten in terms of four partial summations as:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{nk}, \ k = 0, 1, ..., N-1 \\ &= \sum_{m=0}^{\frac{N}{4}-1} x(4m) W_N^{k(4m)} + \sum_{m=0}^{\frac{N}{4}-1} x(4m+1) W_N^{k(4m+1)} \end{aligned}$$

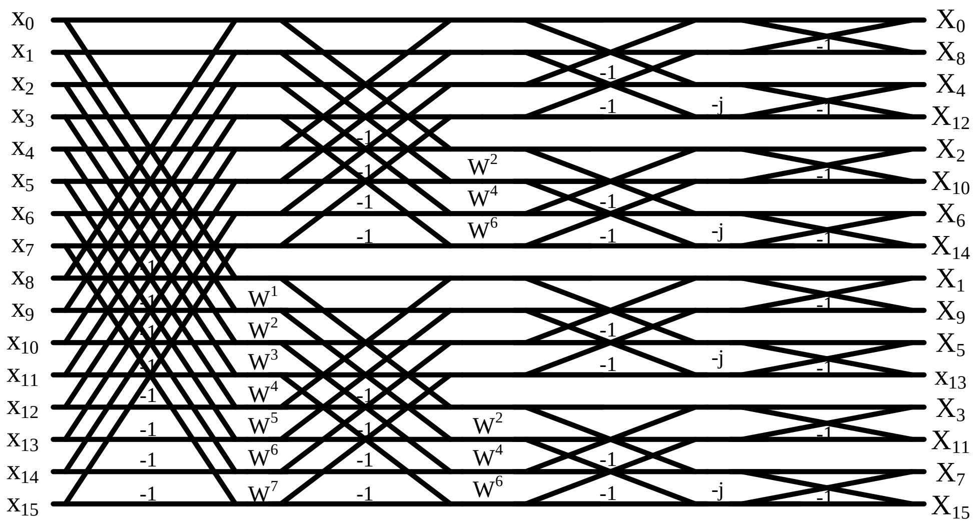


Figure 2.3: The signal flow diagram of 16-point radix-2 DIF algorithm.

$$
\begin{aligned}
& + \sum_{m=0}^{\frac{N}{4}-1} x(4m+2) W_N^{k(4m+2)} + \sum_{m=0}^{\frac{N}{4}-1} x(4m+3) W_N^{k(4m+3)} \\
= \; & \sum_{m=0}^{\frac{N}{4}-1} x(4m) W_N^{k(4m)} + W_N^k \sum_{m=0}^{\frac{N}{4}-1} x(4m+1) W_N^{k(4m)} \\
& + W_N^{2k} \sum_{m=0}^{\frac{N}{4}-1} x(4m+2) W_N^{k(4m)} + W_N^{3k} \sum_{m=0}^{\frac{N}{4}-1} x(4m+3) W_N^{k(4m)} \qquad (2.19)
\end{aligned}
$$

To partition the original problem set, we first decimate the input samples in time index into four subsequences as expressed from (2.20) to (2.23), where $m = 0, 1, ..., \frac{N}{4} - 1$:

$$x_0(m) = x(4m), \; m = 0, 1, ..., \frac{N}{4} - 1 \qquad (2.20)$$

$$x_1(m) = x(4m+1), \; m = 0, 1, ..., \frac{N}{4} - 1 \qquad (2.21)$$

$$x_2(m) = x(4m+2), \; m = 0, 1, ..., \frac{N}{4} - 1 \qquad (2.22)$$

$$x_3(m) = x(4m+3), \; m = 0, 1, ..., \frac{N}{4} - 1 \qquad (2.23)$$

Thus, the four problem subset of length-$\frac{N}{4}$ DFT can be defined (2.24) to (2.27), where $k = 0, 1, ..., \frac{N}{4} - 1$:

$$X_0(k) = \sum_{m=0}^{\frac{N}{4}-1} x(4m)W_N^{k(4m)} = \sum_{m=0}^{\frac{N}{4}-1} x_0(m)W_{\frac{N}{4}}^{km} \tag{2.24}$$

$$X_1(k) = \sum_{m=0}^{\frac{N}{4}-1} x(4m+1)W_N^{k(4m)} = \sum_{m=0}^{\frac{N}{4}-1} x_1(m)W_{\frac{N}{4}}^{km} \tag{2.25}$$

$$X_2(k) = \sum_{m=0}^{\frac{N}{4}-1} x(4m+2)W_N^{k(4m)} = \sum_{m=0}^{\frac{N}{4}-1} x_2(m)W_{\frac{N}{4}}^{km} \tag{2.26}$$

$$X_3(k) = \sum_{m=0}^{\frac{N}{4}-1} x(4m+3)W_N^{k(4m)} = \sum_{m=0}^{\frac{N}{4}-1} x_3(m)W_{\frac{N}{4}}^{km} \tag{2.27}$$

Note that $X_0(k)$, $X_1(k)$, $X_2(k)$, and $X_3(k)$ have a period of $N/4$, the original output $X(k)$ can be expressed in terms of these four subsequences for $m = 0, 1, ..., \frac{N}{4} - 1$ as shown below:

$$X(k) = X_0(m) + W_N^m X_1(m) + W_N^{2m} X_2(m) + W_N^{3m} X_3(m) \tag{2.28}$$

$$\begin{aligned} X(k+\frac{N}{4}) &= X_0(m) + W_N^{m+\frac{N}{4}} X_1(m) + W_N^{2(m+\frac{N}{4})} X_2(m) + W_N^{3(m+\frac{N}{4})} X_3(m) \\ &= X_0(m) - jW_N^m X_1(m) - W_N^{2m} X_2(m) + jW_N^{3m} X_3(m) \end{aligned} \tag{2.29}$$

$$\begin{aligned} X(k+\frac{N}{2}) &= X_0(m) + W_N^{m+\frac{N}{2}} X_1(m) + W_N^{2(m+\frac{N}{2})} X_2(m) + W_N^{3(m+\frac{N}{2})} X_3(m) \\ &= X_0(m) - jW_N^m X_1(m) + W_N^{2m} X_2(m) - jW_N^{3m} X_3(m) \end{aligned} \tag{2.30}$$

$$\begin{aligned} X(k+\frac{3N}{4}) &= X_0(m) + W_N^{m+\frac{3N}{4}} X_1(m) + W_N^{2(m+\frac{3N}{4})} X_2(m) + W_N^{3(m+\frac{3N}{4})} X_3(m) \\ &= X_0(m) + jW_N^m X_1(m) - W_N^{2m} X_2(m) - jW_N^{3m} X_3(m) \end{aligned} \tag{2.31}$$

As observed from (2.28) to (2.31), the butterfly (BF) of radix-4 FFT involves four components and can be expressed in matrix form as shown in (2.33). In order to further reduce the arithmetic cost of complex additions, we can rewrite (2.33) by two consecutive radix-2 BF stages as shown in (2.34). In terms of number of additions, the former needs 12 additions, while only 8 additions are required for the later. Since the multiplicative complexity of both decompositions remains unchanged, the later is called radix-2^2 FFT algorithm to demonstrate the fact that it actually has the same number of complex multiplication as that of radix-4 FFT algorithms.

$$\begin{bmatrix} X(k) \\ X(k+\frac{N}{4}) \\ X(k+\frac{N}{2}) \\ X(k+\frac{3N}{4}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -j \end{bmatrix} \begin{bmatrix} X_0(k) \\ W_N^k X_1(k) \\ W_N^{2k} X_2(k) \\ W_N^{3k} X_3(k) \end{bmatrix} \tag{2.32}$$

$$\begin{bmatrix} X(k) \\ X(k+\frac{N}{4}) \\ X(k+\frac{N}{2}) \\ X(k+\frac{3N}{4}) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} X_0(k) \\ W_N^k X_1(k) \\ W_N^{2k} X_2(k) \\ W_N^{3k} X_3(k) \end{bmatrix} \tag{2.33}$$

The radix-4 DIF FFT algorithm can be derived from recursively decimating the output vector in frequency into four subsequences as expressed from (2.34) to (2.37), where $m = 0, 1, ..., \frac{N}{4} - 1$:

$$X_0(m) = X(4m),\ m = 0, 1, ..., \frac{N}{4} - 1 \tag{2.34}$$

$$X_1(m) = X(4m+1),\ m = 0, 1, ..., \frac{N}{4} - 1 \tag{2.35}$$

$$X_2(m) = X(4m+2),\ m = 0, 1, ..., \frac{N}{4} - 1 \tag{2.36}$$

$$X_3(m) = X(4m+3),\ m = 0, 1, ..., \frac{N}{4} - 1 \tag{2.37}$$

Therefore, the original DFT computation (2.1) can be represented as:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{nk}, r = 0, 1, ..., N-1 \qquad (2.38) \\ &= \sum_{n=0}^{\frac{N}{4}-1} x(n) W_N^{nk} + \sum_{n=\frac{N}{4}}^{\frac{N}{2}-1} x(n) W_N^{nk} + \sum_{n=\frac{N}{2}}^{\frac{3N}{4}-1} x(n) W_N^{nk} + \sum_{n=\frac{3N}{4}}^{N-1} x(n) W_N^{nk} \\ &= \sum_{n=0}^{\frac{N}{4}-1} x(n) W_N^{nk} + \sum_{n=0}^{\frac{N}{4}-1} x(n+\frac{N}{4}) W_N^{(n+\frac{N}{4})k} \\ &\quad + \sum_{n=0}^{\frac{N}{4}-1} x(n+\frac{N}{2}) W_N^{(n+\frac{N}{2})k} + \sum_{n=0}^{\frac{N}{4}-1} x(n) W_N^{(n+\frac{3N}{4})k} \\ &= \sum_{n=0}^{\frac{N}{4}-1} x(n) W_N^{nk} + W_4^k \sum_{n=0}^{\frac{N}{4}-1} x(n+\frac{N}{4}) W_N^{nk} \end{aligned}$$

$$
\begin{aligned}
&+W_4^{2k}\sum_{n=0}^{\frac{N}{4}-1} x(n+\frac{N}{2})W_N^{nk} + W_4^{3k}\sum_{n=0}^{\frac{N}{4}-1} x(n)W_N^{nk} \\
= &\sum_{n=0}^{\frac{N}{4}-1}[x(n) + x(n+\frac{N}{4})W_4^{k} + x(n+\frac{N}{2})W_4^{2k} + x(n+\frac{3N}{4})W_4^{3k}]W_N^{nk}
\end{aligned}
$$

Furthermore, can we decimate $X(k)$ into four problem subsets as expressed from (2.39) to (2.42).

$$
\begin{aligned}
X_0(m) &= X(4m) \\
&= \sum_{n=0}^{\frac{N}{4}-1}[x(n) + x(n+\frac{N}{4})W_4^{4m} + x(n+\frac{N}{2})W_4^{2\cdot 4m} + x(n+\frac{3N}{4})W_4^{3\cdot 4m}]W_N^{nm} \\
&= \sum_{n=0}^{\frac{N}{4}-1}[x(n) + x(n+\frac{N}{4}) + x(n+\frac{N}{2}) + x(n+\frac{3N}{4})]W_N^{nm} \\
&= \sum_{n=0}^{\frac{N}{4}-1} x_0(n)W_{\frac{N}{4}}^{mn}, m = 0, 1, ..., \frac{N}{4} - 1 \qquad (2.39) \\
X_1(m) &= X(4m+1) \\
&= \sum_{n=0}^{\frac{N}{4}-1}[x(n) + x(n+\frac{N}{4})W_4^{4m+1} \\
&\quad + x(n+\frac{N}{2})W_4^{2\cdot(4m+1)} + x(n+\frac{3N}{4})W_4^{3\cdot(4m+1)}]W_N^{n(4m+1)} \\
&= \sum_{n=0}^{\frac{N}{4}-1}[x(n) - jx(n+\frac{N}{4}) - x(n+\frac{N}{2}) + jx(n+\frac{3N}{4})]W_N^{n}W_{\frac{N}{4}}^{mn} \\
&= \sum_{n=0}^{\frac{N}{4}-1} x_1(n)W_{\frac{N}{4}}^{mn}, m = 0, 1, ..., \frac{N}{4} - 1 \qquad (2.40) \\
X_2(m) &= X(4m+2) \\
&= \sum_{n=0}^{\frac{N}{4}-1}[x(n) + x(n+\frac{N}{4})W_4^{4m+2} \\
&\quad + x(n+\frac{N}{2})W_4^{2\cdot(4m+2)} + x(n+\frac{3N}{4})W_4^{3\cdot(4m+2)}]W_N^{n(4m+2)} \\
&= \sum_{n=0}^{\frac{N}{4}-1}[x(n) - x(n+\frac{N}{4}) + x(n+\frac{N}{2}) - x(n+\frac{3N}{4})]W_N^{2n}W_{\frac{N}{4}}^{mn}
\end{aligned}
$$

$$\begin{aligned}
&= \sum_{n=0}^{\frac{N}{4}-1} x_2(n) W_{\frac{N}{4}}^{mn}, m = 0, 1, ..., \frac{N}{4} - 1 \qquad (2.41)\\
X_3(m) &= X(4m+3)\\
&= \sum_{n=0}^{\frac{N}{4}-1} [x(n) + x(n+\frac{N}{4}) W_4^{4m+3}\\
&\quad + x(n+\frac{N}{2}) W_4^{2\cdot(4m+3)} + x(n+\frac{3N}{4}) W_4^{3\cdot(4m+3)}] W_N^{n(4m+3)}\\
&= \sum_{n=0}^{\frac{N}{4}-1} [x(n) + jx(n+\frac{N}{4}) - x(n+\frac{N}{2}) - jx(n+\frac{3N}{4})] W_N^{3n} W_{\frac{N}{4}}^{mn}\\
&= \sum_{n=0}^{\frac{N}{4}-1} x_3(n) W_{\frac{N}{4}}^{mn}, m = 0, 1, ..., \frac{N}{4} - 1 \qquad (2.42)
\end{aligned}$$

Similarly, to reduce the additive complexity by replacing the radix-4 BF into two radix-2 BF stages, $x_0(n)$, $x_1(n)$, $x_2(n)$, $x_3(n)$ are expressed from (2.43) to (2.46), where $n = 0, 1, ..., \frac{N}{4} - 1$.

$$\begin{aligned}
x_0(n) &= (x(n) + x(n+\frac{N}{2})) + (x(n+\frac{N}{4}) + x(n+\frac{3N}{4}) \qquad (2.43)\\
x_1(n) &= (x(n) - x(n+\frac{N}{2})) - j(x(n+\frac{N}{4}) - x(n+\frac{3N}{4}) \qquad (2.44)\\
x_2(n) &= (x(n) + x(n+\frac{N}{2})) - (x(n+\frac{N}{4}) + x(n+\frac{3N}{4}) \qquad (2.45)\\
x_3(n) &= (x(n) - x(n+\frac{N}{2})) + j(x(n+\frac{N}{4}) - x(n+\frac{3N}{4}) \qquad (2.46)
\end{aligned}$$

Take 16-point FFT as an example, the signal flow diagram of DIT and DIF radix-4 FFT algorithms are illustrated in Figure 2.4 and Figure 2.5 respectively. It is clear that the non-trivial complex multiplications of radix-4 FFT algorithms will only appear after every two butterfly stages. As such, it provides better spatial regularity than radix-2 FFT algorithms, which is beneficial to hardware implementation.

2.2.3 Split-Radix FFT Algorithm

The split-radix FFT is proposed in late 80s to further reduce the multiplicative complexity of radix-2 and radix-4 FFT algorithms. In [13], as with the radix-2 FFT

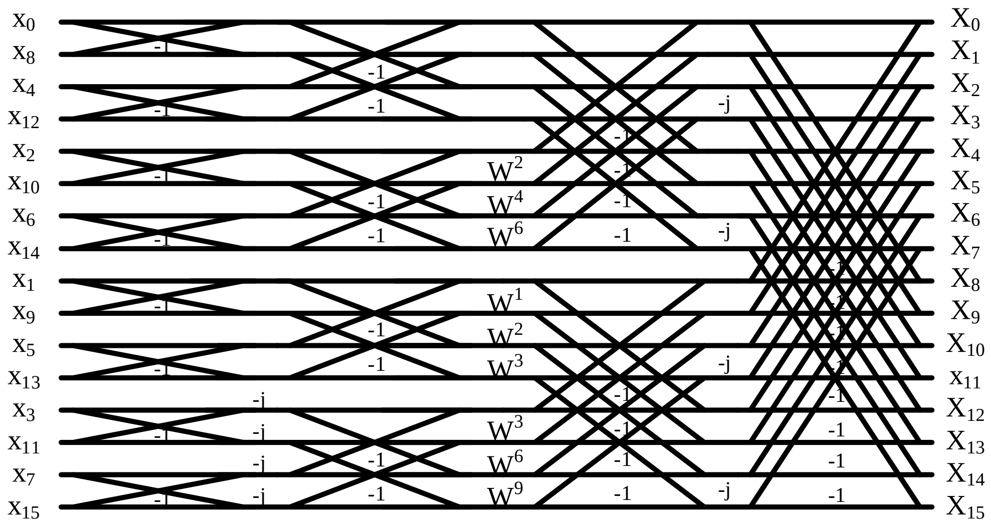


Figure 2.4: The signal flow diagram of 16-point radix-4 DIT algorithm.

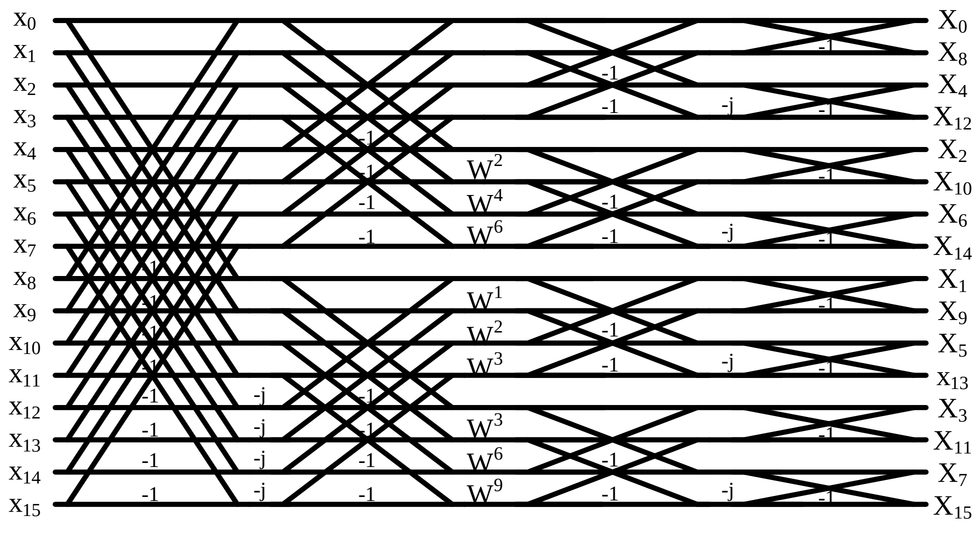


Figure 2.5: The signal flow diagram of 16-point radix-4 DIF algorithm.

case, the DFT operation is first split into odd-half and even-half parts. The odd components are further decomposed into $4k+1$ and $4k+3$ frequency components. Repeating this process for the half- and quarter-length DFTs gives the split-radix FFT algorithm. The split-radix DIT FFT is derived from (2.1) as follows:

$$\begin{aligned}
X(k) &= \sum_{n=0}^{N-1} x(n)W_N^{nk},\ k = 0,1,...,N-1 \\
&= \sum_{m=0}^{\frac{N}{2}-1} x(2m)W_N^{k(2m)} + \sum_{m=0}^{\frac{N}{4}-1} x(4m+1)W_N^{k(4m+1)} + \sum_{m=0}^{\frac{N}{4}-1} x(4m+3)W_N^{k(4m+3)} \\
&= \sum_{m=0}^{\frac{N}{2}-1} x(2m)W_N^{k(2m)} + W_N^k \sum_{m=0}^{\frac{N}{4}-1} x(4m+1)W_N^{k(4m)} \\
&\quad + W_N^{3k} \sum_{m=0}^{\frac{N}{4}-1} x(4m+3)W_N^{k(4m)} \qquad (2.47)
\end{aligned}$$

By decimating the input vector in time index into three subsets as expressed in (2.48) - (2.50), the three problem subsets are defined in (2.51) - (2.53) after applying the identities $W_{\frac{N}{2}} = W_N^2$ and $W_{\frac{N}{4}} = W_N^4$.

$$\begin{aligned}
x_e(n) &= x(2n),\ m = 0,1,...,\frac{N}{2}-1 \qquad (2.48) \\
x_1(n) &= x(4n+1),\ m = 0,1,...,\frac{N}{4}-1 \qquad (2.49) \\
x_3(n) &= x(4n+3),\ m = 0,1,...,\frac{N}{4}-1 \qquad (2.50)
\end{aligned}$$

$$\begin{aligned}
X_e(k) &= \sum_{m=0}^{\frac{N}{2}-1} x(2m)W_{\frac{N}{2}}^{k(2m)} = \sum_{m=0}^{\frac{N}{2}-1} x_e(m)W_{\frac{N}{2}}^{km} \qquad (2.51) \\
X_1(k) &= \sum_{m=0}^{\frac{N}{4}-1} x(4m+1)W_{\frac{N}{4}}^{k(4m)} = \sum_{m=0}^{\frac{N}{2}-1} x_1(m)W_{\frac{N}{4}}^{km} \qquad (2.52) \\
X_3(k) &= \sum_{m=0}^{\frac{N}{4}-1} x(4m+3)W_{\frac{N}{4}}^{k(4m)} = \sum_{m=0}^{\frac{N}{2}-1} x_3(m)W_{\frac{N}{4}}^{km} \qquad (2.53)
\end{aligned}$$

After these three problem subsets are solved by the split-radix algorithm recursively, the solution to the original problem of size N can be obtained according to (2.47) for $k = 0,1,...,N-1$.

A split-radix DIF FFT algorithm can be derived by recursively partitioning the original output samples into three problem subsets in frequency index in a similar fashion. The derivation begins with the definition of DFT expressed in (2.1). Using the results from radix-2 DIF algorithm and radix-4 DIF algorithm, we define $X_e(k) = X(2k)$, $X_1(k) = X(4k+1)$, and $X_3(k) = X(4k+3)$ as follows:

$$\begin{aligned}
X_e(m) &= X(2m) = \sum_{n=0}^{\frac{N}{2}-1}(x(m) + x(m+\frac{N}{2}))W_{\frac{N}{2}}^{mn} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x_e(m)W_{\frac{N}{2}}^{mn},\ m = 0,1,...,,\frac{N}{2}-1 \qquad (2.54) \\
X_1(m) &= X(4m+1) \\
&= \sum_{n=0}^{\frac{N}{4}-1}[x(n) + x(n+\frac{N}{4})W_4^{4m+1} \\
&\quad +x(n+\frac{N}{2})W_4^{2\cdot(4m+1)} + x(n+\frac{3N}{4})W_4^{3\cdot(4m+1)}]W_N^{n(4m+1)} \\
&= \sum_{n=0}^{\frac{N}{4}-1}[x(n) - jx(n+\frac{N}{4}) - x(n+\frac{N}{2}) + jx(n+\frac{3N}{4})]W_N^{n}W_{\frac{N}{4}}^{mn} \\
&= \sum_{n=0}^{\frac{N}{4}-1} x_1(n)W_{\frac{N}{4}}^{mn}, m = 0,1,...,\frac{N}{4}-1 \qquad (2.55) \\
X_3(m) &= X(4m+3) \\
&= \sum_{n=0}^{\frac{N}{4}-1}[x(n) + x(n+\frac{N}{4})W_4^{4m+3} \\
&\quad +x(n+\frac{N}{2})W_4^{2\cdot(4m+3)} + x(n+\frac{3N}{4})W_4^{3\cdot(4m+3)}]W_N^{n(4m+3)} \\
&= \sum_{n=0}^{\frac{N}{4}-1}[x(n) + jx(n+\frac{N}{4}) - x(n+\frac{N}{2}) - jx(n+\frac{3N}{4})]W_N^{3n}W_{\frac{N}{4}}^{mn} \\
&= \sum_{n=0}^{\frac{N}{4}-1} x_3(n)W_{\frac{N}{4}}^{mn}, m = 0,1,...,\frac{N}{4}-1 \qquad (2.56)
\end{aligned}$$

By repeating (2.54), (2.55) and (2.56), we can obtain the solution of original DFT by assembling the results from these three problem subsets. The signal flow diagram of split-radix DIT algorithm is shown in Figure 2.6. Similarly, the signal flow of split-radix DIF algorithm is also given in Figure 2.7. Apparently, these

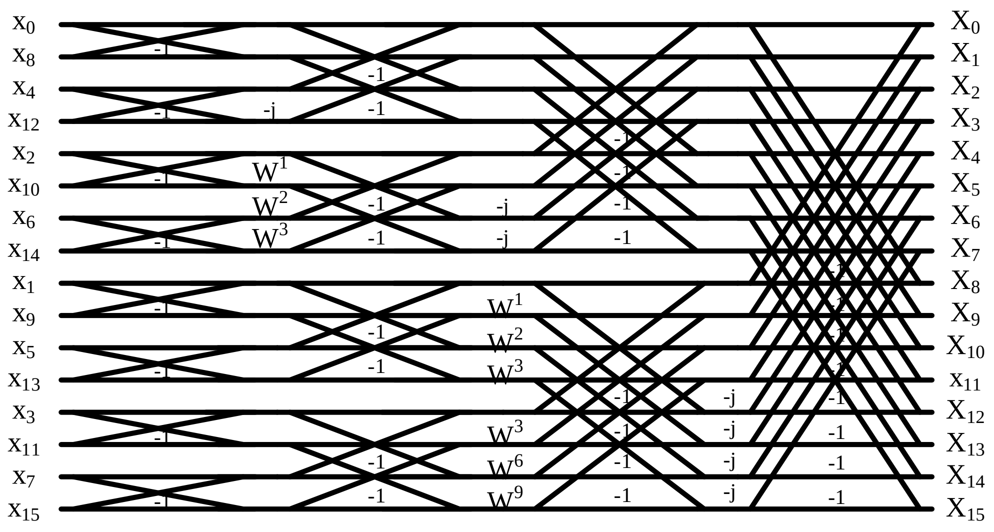


Figure 2.6: The signal flow diagram of 16-point split-radix DIT algorithm.

two schemes have the same multiplicative complexity but the order of complex multiplications is reversed.

2.2.4 Mixed-Radix FFT Algorithm

A larger FFT can be decomposed into cascaded FFT stages with smaller length. Assume the N-point FFT is partitioned into α stages and define

$$N_\beta = \frac{N}{r_1 \cdot r_2 \cdots r_\beta} \quad where \quad 1 \le \beta \le \alpha - 1, \tag{2.57}$$

the final stage is derived as [39]:

$$X(r_1 r_2 \cdots r_{\alpha-1} m_\alpha + r_1 r_2 \cdots r_{\alpha-2} m_{\alpha-1} + \cdots + r_1 m_2 + m_1) \\ = \sum_{q_{\alpha-1}=0}^{r_\alpha - 1} x_{\alpha-1}(q_{\alpha-1}, m_{\alpha-1}) W_{r_\alpha}^{q_{\alpha-1} m_\alpha}. \tag{2.58}$$

The intermediate β-th stage, $x_\beta(q_\beta, m_\beta)$ is given by the recursive equation as:

$$x_\beta(q_\beta, m_\beta) = W_{N_{\beta-1}}^{q_\beta m_\beta} \cdot \sum_{p=0}^{r_\beta - 1} x_{\beta-1}(N_\beta p + q_\beta, m_{\beta-1}) W_{r_\beta}^{p m_\beta}, \tag{2.59}$$

where $2 \le \beta \le \alpha - 1$, $0 \le m_i \le r_i - 1$, $0 \le q_i \le N_i - 1$, and $2 \le i \le \alpha$. Each summation represents a r_β-point FFT computation. Please note the above decomposition procedure is not unique. There are various equivalent decompositions for

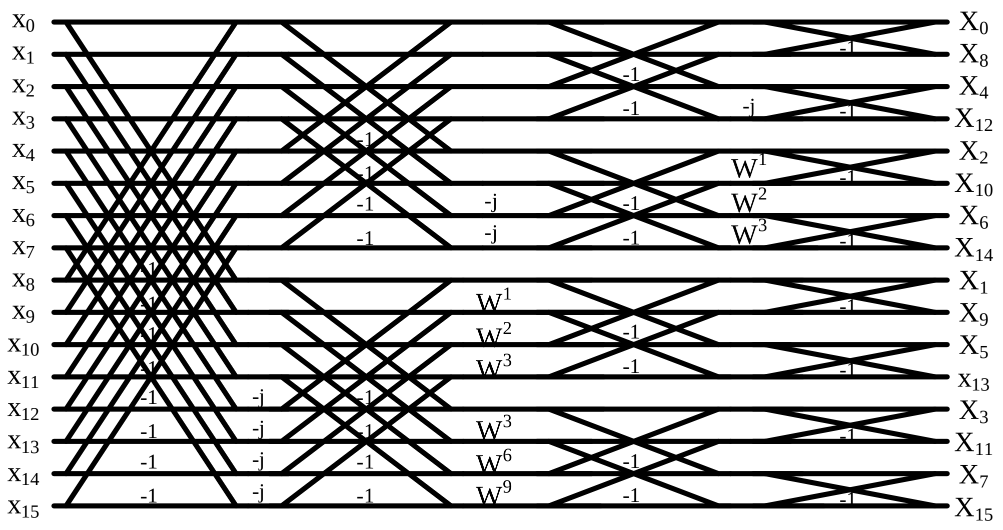


Figure 2.7: The signal flow diagram of 16-point split-radix DIF algorithm.

the same N-point FFT. As a result, many designs of memory-based FFT architecture will decompose a larger FFT computation into several cascaded smaller FFTs and utilize a single FFT core to reduce the hardware cost [40].

2.2.5 Comparison of Different FFT Algorithms

The multiplicative complexity comparison among different FFT algorithms is listed in Table 2.1. In terms of the number of non-trivial multiplications, the split-radix FFT algorithm outperforms the other two because more "$-j$" terms are extracted. However, the irregularity caused by the L-shaped butterfly makes it harder to design a pipeline architecture. Moreover, because the non-trivial multiplications could appear in two successive stages, the required number of multipliers increases. For example, the signal dataflow graphs for a 16-point FFT are illustrated in Fig. 2.4 and Fig. 2.6 for the radix-4 FFT and the split-radix FFT respectively. Obviously, for designing a pipeline architecture, the radix-4 FFT only requires one set of complex multipliers, while the split-radix FFT requires two sets of complex multipliers. Although some effort has been made [27] to share the complex multipliers between adjacent BF stages, it inevitably results in additional overhead of control logic.

Table 2.1: The number of non-trivial complex multiplications

FFT Size	Radix-2	Radix-4	Split-Radix
8	2	2	2
16	10	8	8
32	34	28	26
64	98	76	72
128	258	204	186
256	642	492	456
512	1538	1196	1082
1024	3586	2732	2504
2048	8194	6316	5690
4096	18434	13996	12744

2.3 Implementation of Complex Multiplier

Considering the actual hardware implementation, the accuracy of FFT module is an important design factor of system performance. In practice, fixed-point arithmetic is used to implement FFT in hardware because it is not possible to keep infinite resolution of coefficients and operations. All coefficients and input signals have to be presented with finite number of bits in binary format depending on the tradeoff between the hardware cost (memory usage) and the accuracy of output signals. Regardless of which FFT algorithm is adopted, the complex multiplier can be realized by different approaches. Besides the direct implementation of the complex multiplier, the numerical decomposition [41], CORDIC algorithm and IntFFT [42] are also utilized for the design of FFT architecture and have different advantages. For example, the numerical decomposition consists of only three multiplications and three additions with one additional coefficient look-up table. CORDIC algorithm rewrites the conventional complex multiplication with a series of angle rotations. It is particularly useful when no multiplier unit is available. During the past decade, lifting factorization has been developed as a useful tool for the construction of wavelets [43]. In [42], the idea of lifting scheme is further applied to compute the non-trivial complex multiplication where the original operation is expressed as three lifting steps. In this section, different implementations of complex multiplier are introduced respectively.

2.3.1 Numerical Decomposition

Let $t = c + js$ be a complex number with magnitude one (i.e. $|t| = 1$) and $x = x_r + jx_i$. The direct implementation of complex multiplications requires four real multiplications and 2 additions. The numerical decomposition expressed in (2.60) and (2.61) rewrites the original complex multiplication with three multiplications, three additions and three constants [41]. In brief, one multiplication is replaced by one addition. However, two additional coefficient tables are required since both $c + s$ and $c - s$ are assumed to be pre-calculated.

$$cx_r - sx_i = x_r(c - s) + s(x_r - x_i) \tag{2.60}$$

$$cx_i + sx_r = x_i(c+s) + s(x_r - x_i). \tag{2.61}$$

2.3.2 CORDIC-Based Complex Multiplier

The CORDIC algorithm is a simple and efficient algorithm to calculate hyperbolic and trigonometric functions. As shown in Figure 2.8, the basic idea of CORDIC algorithm is to complete the angle rotation in the polar plane with a series of smaller incremental rotations. Therefore the desired rotation angle, Θ, is decomposed into pre-defined elementary angles as:

$$\Theta = \sum_{n=0}^{N_c-1} \mu_n a_n + \epsilon, \tag{2.62}$$

where N_c is the number of total elementary angles, $\mu_i \in \{1, -1\}$ is used to indicate the direction of angle rotations, a_i is the ith elementary angle, and ϵ is the difference between the real angle and the approximated angle. It has been extended and utilized for FFT implementations to perform complex multiplications with only additions and shifts. By calculating the product of all cosine values in advance, each iteration only involves one addition and one subtraction. In other words, the complex multiplication is represented as

$$\begin{aligned} y &= [x_r \quad x_i] \prod_{n}^{Nc} \begin{bmatrix} \cos a_n & \sin a_n \\ -\sin a_n & \cos a_n \end{bmatrix} \\ &= \underbrace{(\prod_{n}^{N_c} \cos a_n)}_{pre-calculated} [x_r \quad x_i] \prod_{n}^{N_c} \begin{bmatrix} 1 & \tan a_n \\ -\tan a_n & 1 \end{bmatrix}. \end{aligned} \tag{2.63}$$

From the viewpoint of hardware design, the advantage of CORDIC is that only additions and shifts are required. It is very suitable for processor-based designs where multiplier unit is not available. It also leads to smaller hardware cost for the ASIC designs because no general multiplier is adopted. Additionally, when implemented in soft or dedicated hardware the algorithm is suitable for pipelining. However, it inevitably degrades the output accuracy since it is an approximation of the original complex multiplications.

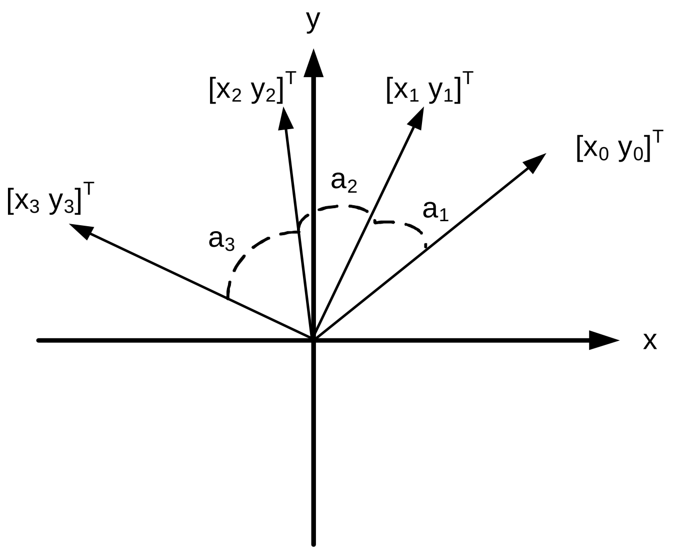


Figure 2.8: The operation of CORDIC algorithm.

2.3.3 LS-Based Complex Multiplier

IntFFT [42] is an integer-to-integer mapping transform in which the wavelet transform concept of lifting scheme is applied to FFT algorithms. Originally, the lifting scheme was developed as a useful tool for constructing biorthogonal wavelets and perfect reconstruction (PR) filter banks [43]. Since it only involves integer operations, lossless transform is possible. The main idea of IntFFT is to utilize the lifting scheme in the conventional FFT by decomposing the original complex multiplications into three lifting steps [42]. All the original twiddle factors in the FFT computation are presented in lifting coefficient format. Let $t = c + js$ be a complex number with magnitude one (i.e. $|t| = 1$) and $x = x_r + jx_i$. Then the original complex multiplication can be represented in matrix form as [42]:

$$\begin{aligned} tx &= (cx_r - sx_i) + j(cx_i + sx_r) \\ &= \begin{bmatrix} 1 & j \end{bmatrix} \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} x_r \\ x_i \end{bmatrix} = \begin{bmatrix} 1 & j \end{bmatrix} R \begin{bmatrix} x_r \\ x_i \end{bmatrix} \end{aligned} \tag{2.64}$$

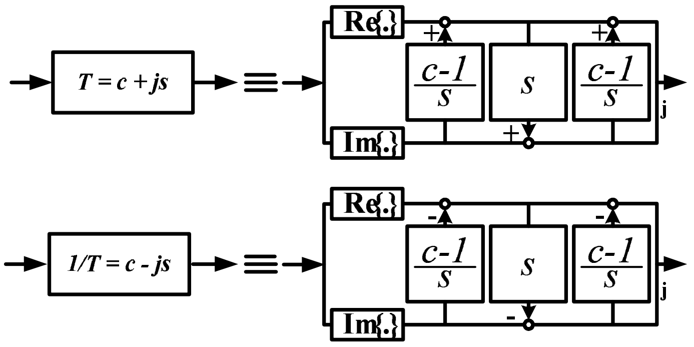


Figure 2.9: Representation of twiddle factors as three lifting steps.

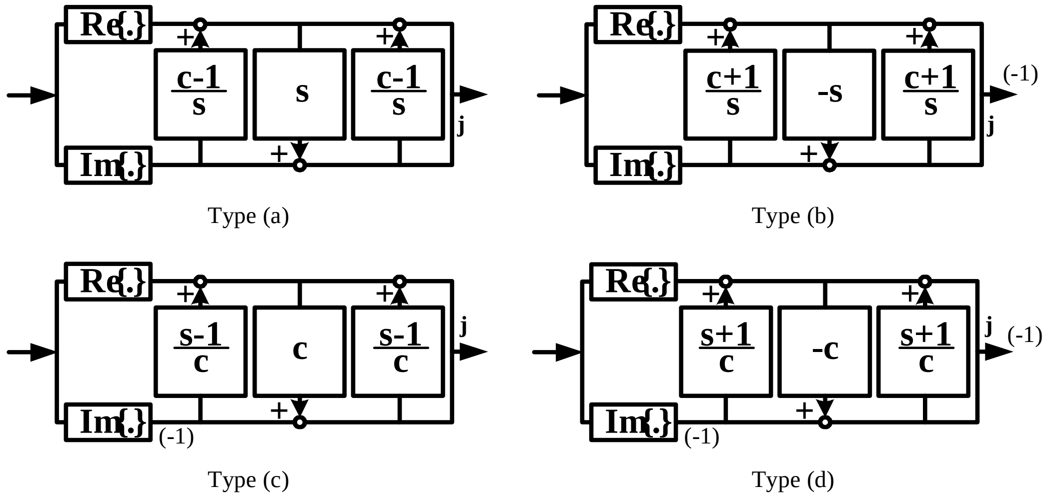


Figure 2.10: Four equivalent decompositions of lifting scheme.

Furthermore, the R matrix shown in Eq. (2.64) can be decomposed into three lifting steps [43].

$$R = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} = \begin{bmatrix} 1 & \frac{c-1}{s} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{c-1}{s} \\ 0 & 1 \end{bmatrix} \tag{2.65}$$

In this equation, s and c represent $\sin\theta$ and $\cos\theta$ respectively, where θ is the angle of twiddle factors as defined in Eq. (2.2). As a result, the original complex multiplications of multiplying twiddle factors can be represented in lifting scheme form as shown in Fig. 2.9. Recall that no matter what operation is performed in the lifting steps, we can reconstruct the original input without any distortion if the same operation is performed again in the corresponding inverse lifting steps.

Table 2.2: Four equivalent decompositions of lifting scheme.

	Type (a)	Type (b)	Type (c)	Type (d)
P	$P_a = \frac{c-s}{s}$	$P_b = \frac{c+1}{s}$	$P_c = \frac{s-1}{c}$	$P_d = \frac{s+1}{c}$
Q	$Q_a = s$	$Q_b = -s$	$Q_c = c$	$Q_d = -c$

In other words, IntFFT is able to produce lossless output. As shown in Fig. 2.9, the sign change inverse (SCI) property is also illustrated. It leads to an easy implementation of the inverse transform by replacing additions in the lifting paths with subtractions. Notice when the angle of the twiddle factors, θ_i, goes smaller, the value of $p_i[n]$ in the above decomposition will become infinitely large. Different decompositions of lifting scheme are derived to avoid this problem. Four alternative types of lifting decompositions are illustrated in Figre 2.10 [42] and summarized in Table 2.2. Because these four possible different types of lifting decompositions can be used interchangeably, it is of interests to analyze the performance of each lifting decompositions. The MSE analysis of different implementations of complex multipliers is derived in Chapter 3 to give a quantitative comparison.

2.4 Applications of FFT

In this section, some common FFT applications are introduced to demonstrate the importance of FFT algorithms.

2.4.1 Fast Convolution

The FFT algorithms provides a convenient way to perform fast convolutions without computing the convolution sum. Assuming two length-N complex input $x(n)$ and $y(n)$ sequences, the fast convolution of $x(n)$ and $y(n)$ can be calculated using the following procedure:

- Step 1: Complete length-N FFT computation of $x(n)$ and $y(n)$, named $X(k)$ and $Y(k)$ respectively.

- Step 2: Multiply the FFT results to form the product: $Z(k) = X(k)Y(k)$.
- Step 3: Complete length-N IFFT of $Z(k)$.

The direct computation of convolution needs N^2 complex multiplications. For example, given two sequences with 16 samples, the direct convolution requires 256 complex multiplications while the FFT-based fast convolution only requires 46 complex multiplications as listed in Table 2.1.

2.4.2 OFDM Systems

OFDM is a digital multi-carrier modulation scheme, which uses a large number of orthogonal sub-carriers. Each sub-carrier is modulated with a conventional modulation scheme at a low symbol rate, maintaining data rates similar to conventional single-carrier modulation schemes in the same bandwidth. In practice, OFDM signals are generated using the FFT algorithm. The main advantage of OFDM over single-carrier schemes is its ability to cope with severe channel conditions. The sub-carrier frequencies are chosen so that the sub-carriers are orthogonal to each other, meaning that cross-talk between the sub-channels is eliminated and inter-carrier guard bands are not required.

One key principle of OFDM is that since low symbol rate modulation schemes suffer less from intersymbol interference caused by multipath, it is advantageous to transmit a number of low-rate streams in parallel instead of a single high-rate stream. Since the duration of each symbol is long, it is feasible to insert a guard interval between the OFDM symbols, thus eliminating the intersymbol interference. The cyclic prefix, which is transmitted during the guard interval, consists of the end of the OFDM symbol copied into the guard interval, and the guard interval is transmitted followed by the OFDM symbol. The reason that the guard interval consists of a copy of the end of the OFDM symbol is so that the receiver will integrate over an integer number of sinusoid cycles for each of the multipaths when it performs OFDM demodulation with the FFT. Besides, since the effects of frequency-selective channel conditions can be considered as flat over an OFDM sub-channel if the sub-channel is sufficiently narrow-banded, it makes

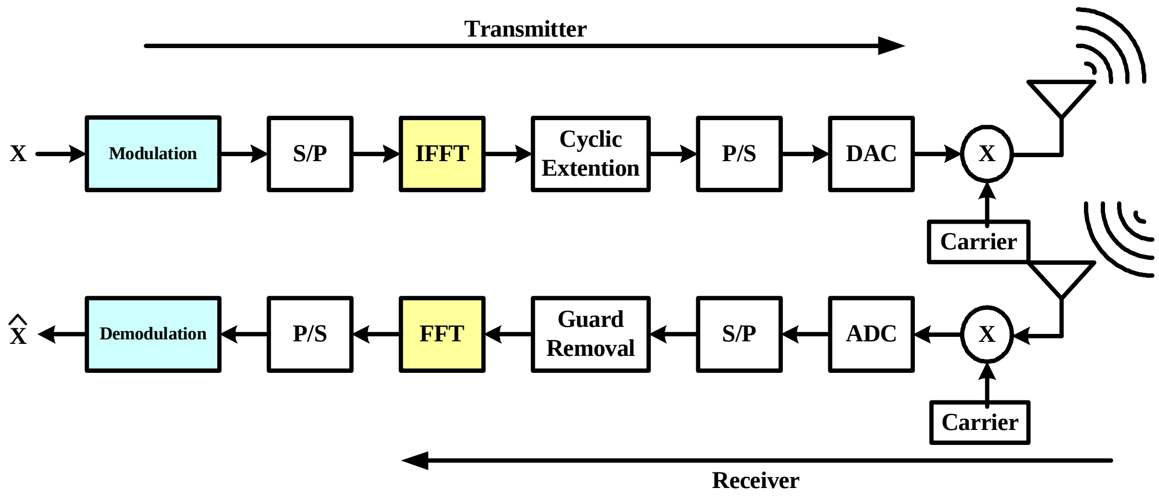


Figure 2.11: A simplified block diagram of OFDM systems.

equalization much simpler at the receiver in OFDM in comparison to conventional single-carrier modulation. A conceptual diagram of OFDM systems is illustrated in Figure 2.11.

3

Integer FFT with Optimized Coefficient Set

In this chapter, the noise model of arithmetic rounding errors is first introduced. In order to improve the overall SQNR performance of FFT computation, the principle of finding the optimized coefficient set of IntFFT is proposed. IntFFT has been regarded as an approximation of original FFT since it utilizes LS and decomposes the complex multiplication of twiddle factor into three lifting steps. Based on the observation of the quantization loss model of lifting operations, we can select an optimized coefficient set and achieve better SQNR. A mixed-radix 128-point FFT is used to compare the SQNR performance between IntFFT and other FFT implementations.

3.1 Noise Model of Quantization Loss

There are two fundamental numbering systems for representing numbers in a digital system: fixed point and floating point. The dynamic range of floating point numbering system is much higher than that of fixed point numbering system. However, there is always a tradeoff between these two systems because the cost of floating point operations is higher than the cost of the other.

In binary representation of a real number, x, with $B+1$ bits, there are commonly three formats: sign magnitude, one's complement and two's complement, in which two's complement is the most popular one. The only difference between these three representations is how the negative numbers are represented. In the two's complement format, a real number is represented as:

$$x = X_m(-b_0 + \sum_{i=1}^{B} b_i 2^{-i}). \tag{3.1}$$

Note that with $B + 1$ bits, the smallest distinguishable difference between two quantized numbers, the resolution of selected numbering system, is:

$$\Delta = X_m 2^{-B}, \tag{3.2}$$

and all quantized numbers lie on the range $-X_m \leq x \leq X_m$.

In performing arithmetic operations in digital computers with fixed-point numbers, it is necessary to quantize numbers by either truncation or rounding due to the finite resources of internal memory storage. For example, multiplying two 16-bit numbers yields a product with up to 32-bit precision. In general, the product will be quantized back to 16 bits. Both truncation and rounding will introduce a quantization error. The additive noise model of quantization loss is widely adopted to measure the effect of the fixed length operations in DSP systems [11, 17]. The quantized product can be expressed as the sum of unquantized product and an uniformly-distributed additive quantization noise. Consider the multiplication of quantized numbers $\hat{x}$ and $\hat{a}$, the product y itself will be quantized to a $(B+1)$-bit number, $\hat{y} = Q_B[y]$. The quantized product term can be expressed as the unquantized product with an additive quantization noise source, e, as depicted in Fig. 3.1. For the rounding noise with two's complement numbers, assuming that e is an uniformly distributed random variable whose probability density function (pdf) is shown in (3.3), its variance can be easily calculated as $\sigma^2 = \frac{\Delta^2}{12}$, where $\Delta = 2^{-B}$,

$$p(e) = \begin{cases} \frac{1}{\Delta}, & -\frac{\Delta}{2} < e \leq \frac{\Delta}{2} \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

On the other hand, the pdf of truncation error is always negative and falls in the range expressed in (3.4).

$$p(e) = \begin{cases} \frac{1}{\Delta}, & -\Delta < e \leq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

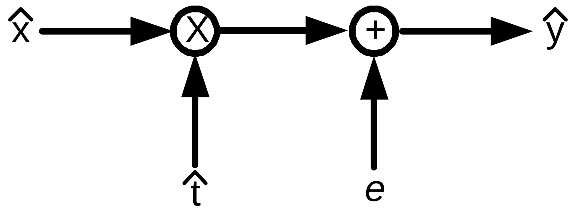


Figure 3.1: Additive noise model of quantization loss.

3.2 MSE of Various Complex Multipliers

For a conventional complex multiplier, assume that $x = x_r + jx_i$ represents the input sample, and that $t = c + js$ represents the complex twiddle factor. Then the quantized output is represented as follows:

$$Q(D) = [Q(cx_r) - Q(sx_i)] + j[Q(cx_i) + Q(sx_r)]. \tag{3.5}$$

Consequently, the overall quantization loss and the variance of quantization loss are expressed as

$$\begin{aligned} Q(D) - D &= (e_1 - e_2) + j(e_3 + e_4) \\ E_C[|Q(D) - D|^2] &= E[(e_1 - e_2)^2 + (e_3 + e_4)^2] \\ &= E[e_1^2] + E[e_2^2] + E[e_3^2] + E[e_4^2] \\ &= 4\sigma^2 \end{aligned}$$

where $e_1 = Q(cx_r) - cx_r$, $e_2 = Q(sx_i) - sx_i$, $e_3 = Q(cx_i) - cx_i$, and $e_4 = Q(sx_r) - sx_r$ are all real-valued random variables.

The numerical decomposition with quantization loss is depicted in Fig. 3.2. Three independent real-valued uniform random variables are introduced to model the quantization loss. Without loss of generality, we assume there is no loss from the pre-calculated coefficients, $c + s$ and $c - s$, Therefore, the MSE of numerical decomposition is expressed as:

$$\begin{aligned} E_N[|Q(Y) - Y|^2] &= E[|(e_0 + e_1) + j(e_2 + e_1)|^2] \\ &= E[e_0^2] + 2E[e_1^2] + E[e_2^2] = 4\sigma^2 \end{aligned} \tag{3.6}$$

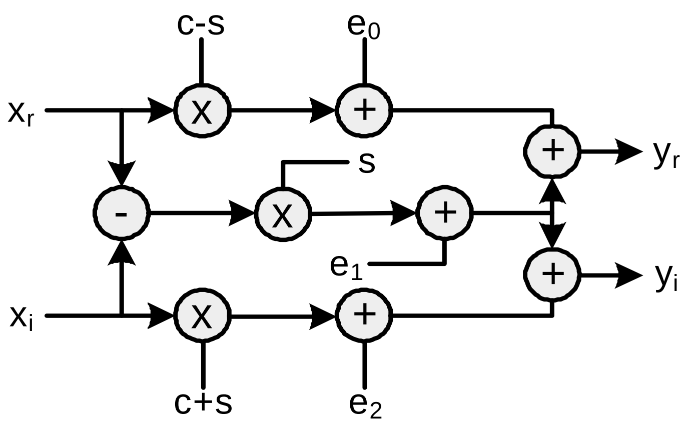


Figure 3.2: The quantization loss model of the numerical decomposition.

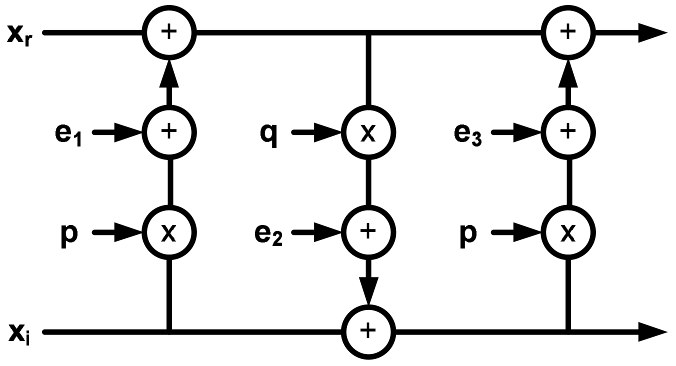


Figure 3.3: The equivalent quantization loss model of LS-based complex multiplier.

For the LS-based complex multipliers depicted in Fig. 3.3, the lifting coefficients are generated from corresponding twiddle factors by $p[n]$ and $q[n]$ where both $p[n]$ and $q[n]$ are real-valued discrete functions and can be expressed as:

$$\begin{aligned} p[n] &= \frac{\cos(\theta)-1}{\sin(\theta)} \quad (3.7)\\ q[n] &= \sin(\theta) \quad (3.8)\end{aligned}$$

where $\theta = \frac{-2\pi n}{N}$, and n is the index of the twiddle factors. Again, we assume all random variables are all statistically independent and uniformly distributed. The variance of the quantization loss, $f_L[n]$, is derived in (3.11).

$$\begin{aligned} D &= [(1+p[n]q[n])x_r + p[n](2+p[n]q[n])x_i] \\ &\quad +j[(1+p[n]q[n])x_i + q[n]x_r] \quad (3.9)\\ Q(D) &= [(1+p[n]q[n])x_r + p[n](2+p[n]q[n])x_i + (1+p[n]q[n])e_1 + p[n]e_2 + e_3] \\ &\quad +j[(1+p[n]q[n])x_i + q[n]x_r + q[n]e_1 + e_2] \quad (3.10)\end{aligned}$$

$$\begin{aligned} E_{LS}[|Q(D)-D|^2] &= E[[(1+p[n]q[n])e_1 + p[n]e_2 + e_3]^2 + (q[n]e_1 + e_2)^2] \\ &= [(\frac{\cos(\omega n)-1}{\sin(\omega n)})^2 + 3]\cdot\sigma^2 = f_L[n]\sigma^2 \quad (3.11)\end{aligned}$$

3.3 Periodicity Property of Lifting Coefficients

Since lifting coefficients are the functions of sinusoids, we use the periodicity property of sinusoids to present lifting coefficients with a smaller set. For example, if $\frac{N}{8} < n < \frac{N}{4}$, $P_d[n]$ and $Q_d[n]$ can be represented by $P_a[n]$ and $Q_a[n]$ as follows:

$$\begin{aligned} P_d[\frac{N}{4}-n] &= \frac{\sin(-\frac{2\pi}{N}(\frac{N}{4}-n))+1}{\cos(-\frac{2\pi}{N}(\frac{N}{4}-n))} = P_a[n] \quad (3.12)\\ Q_d[\frac{N}{4}-n] &= -\cos(-\frac{2\pi}{N}(\frac{N}{4}-n)) = Q_a[n], \quad (3.13)\end{aligned}$$

Similarly, if $\frac{N}{2} \le n_i \le \frac{5N}{8}$, $P_b[n]$ and $Q_b[n]$ can be represented by $P_a[n]$ and $Q_a[n]$ as follows:

$$\begin{aligned} P_b[\frac{N}{2}+n_i] &= \frac{\cos(-\frac{2\pi}{N}(\frac{N}{2}+n_i))+1}{\sin(-\frac{2\pi}{N}(\frac{N}{4}+n_i))} = \frac{-\cos\theta_i+1}{-\sin\theta_i} = P_a[n_i] \quad (3.14)\\ Q_b[\frac{N}{2}+n_i] &= -(-\sin\theta_i) = Q_a[n_i] \quad (3.15)\end{aligned}$$

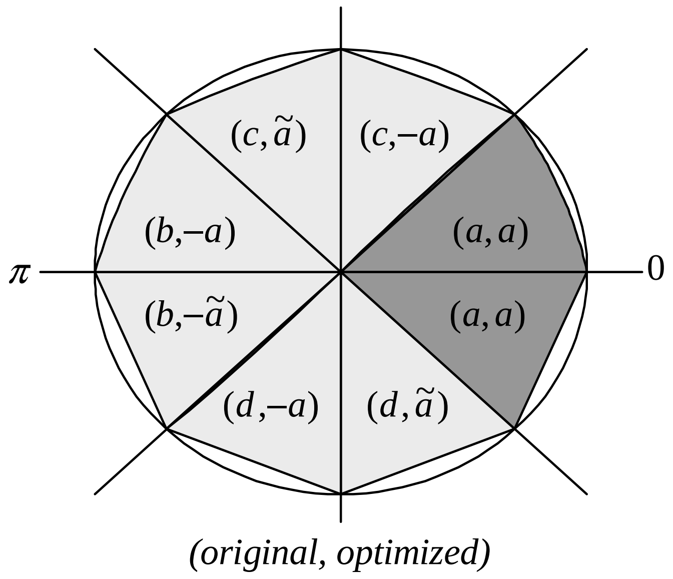


Figure 3.4: The optimized coefficient set of IntFFT.

Therefore, the polar plane can be partitioned into 8 sub-regions. Fig. 3.4 shows the mapping diagram of each decomposition in different sub-regions, where $-a$ and $\tilde{a}$ indicate sign reverse and index reverse respectively. All the other decompositions can be represented by Type (a). For instance, the non-trivial twiddle factors used in the first multiplier stage of 128-point radix-4 DIF FFT is mapped into Type (a) as shown in table 3.1. Combining the idea of coefficient selection from previous section, the criteria of choosing lifting coefficient must fulfill two conditions:

1) The dynamic range of selected coefficients should be smaller to keep lower MSE.

2) The selected coefficient set must be able to cover all other sub-regions.

Although the selection of coefficient set is not unique, it is convenient to choose the first sixteen pairs of Pa and Qa as the optimized coefficient set. The theoretical MSE performance comparison of different schemes is shown in Fig. 3.5. It is obvious that the LS-based complex multiplier with optimized coefficient set has the smallest MSE compared to the other methods.

Table 3.1: The mapping of non-trivial multiplications into the optimized set.

Original Index	Original Type	Mapped Index	Mapped Type
1	(a)	1	(a)
2	(a)	2	(a)
3	(a)	3	(a)
4	(a)	4	(a)
17	(d)	15	(a)
18	(d)	14	(a)
19	(d)	13	(a)
20	(d)	12	(a)
49	(b)	15	(a)
50	(b)	14	(a)
97	(c)	1	(a)
98	(c)	2	(a)
...	...	...	...

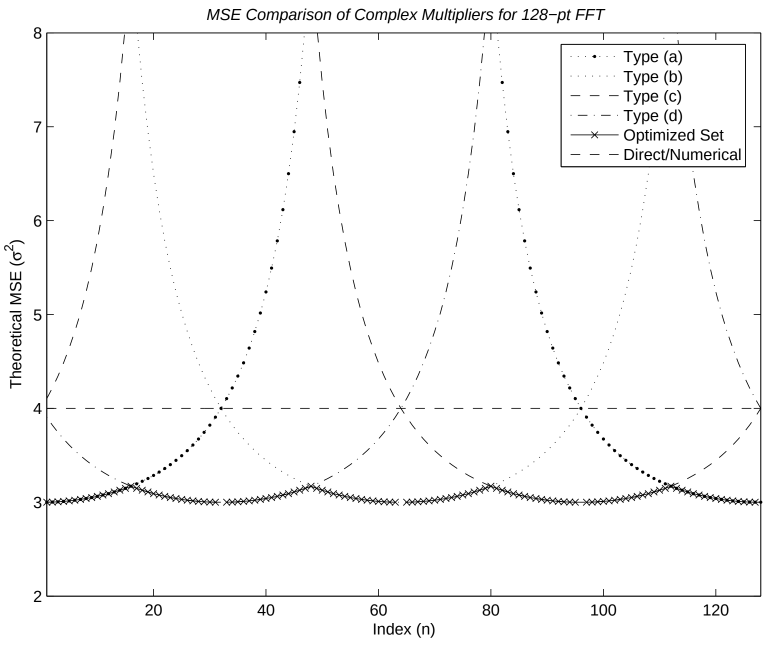


Figure 3.5: MSE comparison chart of different complex multiplier implementations.

3.4 Performance Evaluation of Optimized Coefficient Set

The theoretical performance evaluation of each FFT implementation has been given in previous works. James [21] derived the fixed-point MSE analysis of quantization loss for mixed-radix FFT algorithms with conventional complex multipliers. Perlow and Denk [22] proposed an error propagation model to estimate the performance of finite wordlength FFT architecture. Park and Cho [24] also used a propagation model to derive the error analysis for CORDIC implementations. The comparison of a vector rotation example has been addressed in Table V of [44], which shows that the direct implementation outperforms CORDIC algorithm. In this section, all four kinds of FFT implementations are simulated with fixed-point operations. A mixed-radix 128-point IFFT/FFT with different decomposition schemes is implemented to quantitively compare their SQNR performance. FxpFFT, NumFFT and CORFFT stand for direct implementation, numerical decomposition and CORDIC algorithm respectively. The experiment setup is as follows. First, fixed-point IFFT is performed with random quadrature phase shift

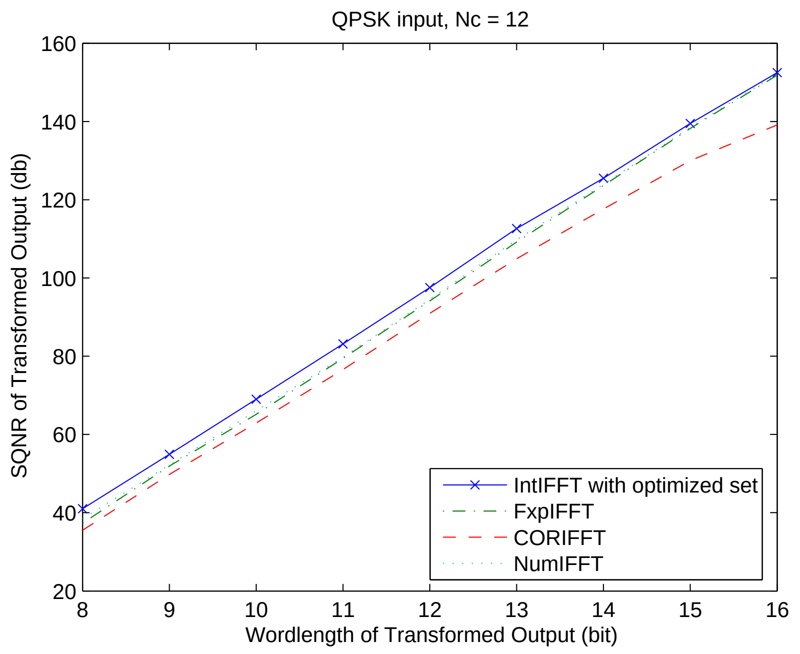


Figure 3.6: SQNR comparisons of IFFT output with QPSK signal input, Nc = 12.

keying (QPSK) signals. 1,000 trials are made for each IFFT. The wordlength of coefficients and the number of iterations used in the CORDIC algorithm is set to 12 and 14. The wordlength of internal variables is swept from 8 bits to 18 bits. As shown in Fig. 3.6 and Figure 3.7, SQNR of IntIFFT with optimized coefficient set has better SQNR performance than others.

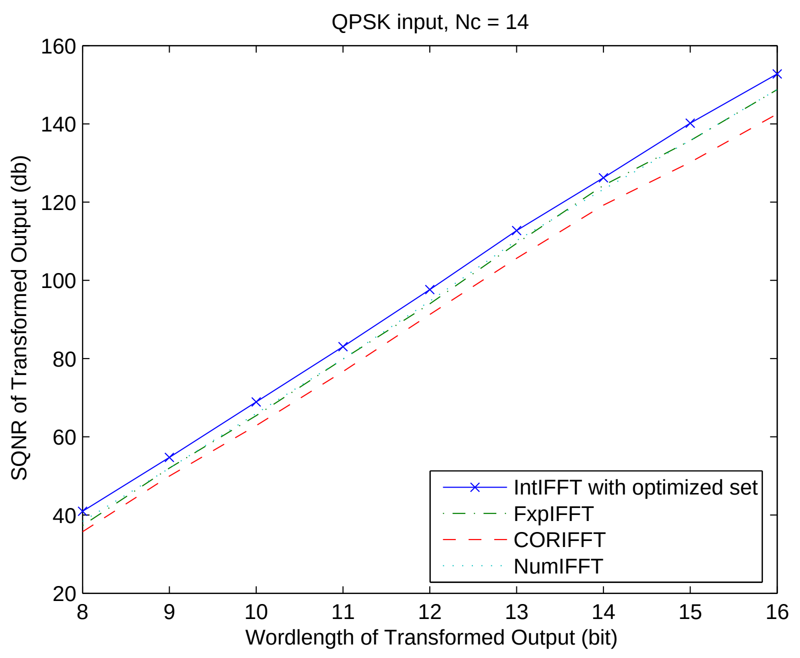


Figure 3.7: SQNR comparisons of IFFT output with QPSK signal input, Nc = 14.

4

Fixed-Point Analysis of FFT Algorithms

The effect of finite wordlength in digital systems has been studied during the past decades. Weistein [45] classified the quantization loss into four categories including quantization of coefficients, error due to A/D conversion, roundoff noise due to arithmetic rounding operations, and the constraint on signal level to maintain the dynamic range and prevent overflow. Oppenheim [17] analyzed the effect of finite precision for digital filtering and radix-2 DIT FFT but did not consider the fact that some twiddle factors are trivial and do not contribute any noise. Thong and Liu [18] also investigated the same phenomenon for both DIT and DIF radix-2 FFT with neglection of the quantization of coefficient because it is shown the error due to the quantization of coefficient is less significant than that of arithmetic rounding operations [45]. However, since split-radix FFT and other higher-radix FFT algorithms are proved to be computationally efficient than radix-2 [13] and are widely used for hardware implementation [27, 46], as such the study of finite precision effect of all FFT algorithms is of increasing significance.

4.1 Matrix Representation of FFT Algorithms

In this section, we derive the equivalent matrix form of both DIF and DIT FFT algorithms. Although the alternative DIT and DIF FFT algorithms have the same multiplicative complexity, the sequence of butterfly stages and twiddle factor stages is reversed. In other words, the signal flow of two alternative representations is actually the mutual mirroring of each other.

4.1.1 Propagation Model of DIF FFT Algorithms

For a N-point DIF FFT computation as shown in Fig. 4.1, define $\alpha = \log_2 N$, we can rewrite DIF FFT in the matrix representation as follows:

$$X_F = \prod_{i=0}^{\alpha-1} w_{F_i} B_{\alpha,\alpha-i} x. \tag{4.1}$$

where each matrix is explained as follows:

1) x is the N-by-1 input vector.

2) X_F is the N-by-1 transformed output vector.

3) $B_{\alpha,\alpha-i}$ is the N-by-N equivalent butterfly matrix at the ith stage of 2^α-point DIF FFT.

4) w_{F_i} is the N-by-N equivalent twiddle factor matrix at the ith stage, where w_{F_i} is a diagonal matrix whose elements are the twiddle factors of the ith stage.

Take an 8-point DIF FFT as example, the corresponding $B_{\alpha,\alpha-i}$ matrices are defined below. The w_{F_i} matrices of the radix-2 DIF FFT algorithms are also listed. Please note the variation of w_{F_i} as the radix of FFT algorithms changes.

$$B_{3,3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$= I_1 \otimes B_{3,3} \tag{4.2}$$

$$B_{3,2} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}$$

$$= I_2 \otimes B_{2,2} \tag{4.3}$$

$$B_{3,1} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

$$= I_4 \otimes B_{1,1} \tag{4.4}$$

$$B_{1,1} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = H_2 \tag{4.5}$$

$$w_{F_0} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & W_8^1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & W_8^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & W_8^3 \end{bmatrix} \tag{4.6}$$

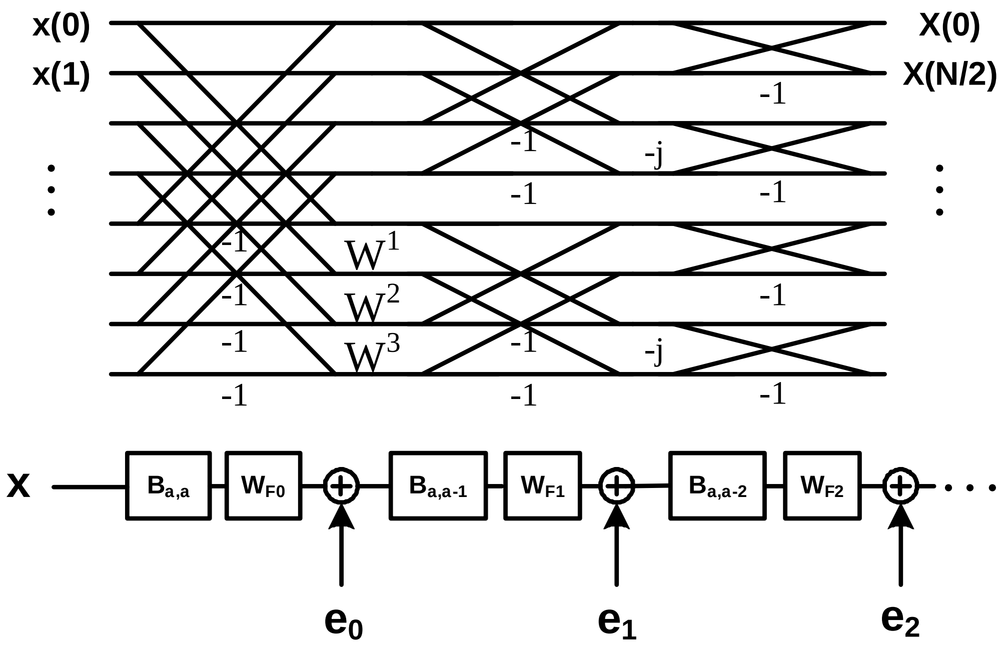


Figure 4.1: Propagation model of quantization loss for DIF FFT.

$$w_{F_1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & W_8^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & W_8^2 \end{bmatrix} \tag{4.7}$$

where H_2 is the 2-by-2 Walsh matrix, $\otimes$ denotes Kronecker product and I_p is the p-by-p identity martrix. From (4.2) to (4.5), the $B_{\alpha,i}$ matrices can be generalized as:

$$B_{i,i} = H_2 \otimes I_{2^{i-1}}, \tag{4.8}$$

$$B_{\alpha,i} = I_{2^{\alpha-i}} \otimes B_{i,i}, \ \alpha \geq i. \tag{4.9}$$

Define e_i $(0 \leq i \leq \alpha)$ as the corresponding N-by-1 additive noise vector of w_{F_i}. Its elements, $e_i(j)$ $(0 \leq j \leq N-1)$, are uncorrelated complex uniform random variables with zero mean and variance equal to σ_c^2, if a non-trivial complex multiplication is presented at $w_{F_{i,j}}$. An example for the second stage of 8-point DIF split-radix FFT is shown in (4.10).

$$e_1 = \left[\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & e_1(5) & 0 & e_1(7) \end{array} \right]^t \tag{4.10}$$

Let x_i be the intermediate result after i stage, $\tilde{x_i}$ is the corresponding erroneous version with quantization loss due to non-trivial complex multiplications and Δx_i is the difference between the original output and the erroneous output. Therefore, all x_i and Δx_i can be expressed as (4.11) - (4.16):

$$\tilde{x_0} = w_{F_0} B_{3,3} x + e_0 \tag{4.11}$$

$$\begin{aligned} \tilde{x_1} &= w_{F_1} B_{3,2} \tilde{x_0} + e_1 \\ &= w_{F_1} B_{3,2} (w_{F_0} B_{3,3} x + e_0) + e1 \end{aligned} \tag{4.12}$$

$$\begin{aligned} \tilde{x_2} &= w_{F_2} B_{3,1} \tilde{x_1} + e_2 \\ &= w_{F_2} B_{3,1} [w_{F_1} B_{3,2} (w_{F_0} B_{3,3} x + e_0) + e1] \\ &\quad + e2 \end{aligned} \tag{4.13}$$

$$\Delta x_0 = e_0 \tag{4.14}$$

$$\Delta x_1 = w_{F_1} B_{3,2} e_0 + e_1 \tag{4.15}$$

$$\Delta x_2 = w_{F_2} B_{3,1} w_{F_1} B_{3,2} e_0 + w_{F_2} B_{3,1} e_1 + e_2 \tag{4.16}$$

Based on the observation of the recursive representation shown from (4.14) - (4.16), we can find the general expression of the overall quantization loss, ΔX_F, as:

$$\Delta X_F = \Delta x_\alpha = \sum_{i=0}^{\alpha} (\prod_{j=i}^{\alpha-1} w_{F_j} B_{\alpha,\alpha-j}) e_i \tag{4.17}$$

4.1.2 Propagation Model of DIT FFT Algorithms

Similarly, as shown in Fig. 4.2, we can define ΔX_T as the overall output error of DIT FFT algorithms:

$$\Delta X_T = \sum_{i=0}^{\alpha} B_{\alpha,\alpha} (\prod_{j=i}^{\alpha-1} w_{T_j} B_{\alpha,j}) (\prod_{k=i}^{0} w_{T_k}) e_i \tag{4.18}$$

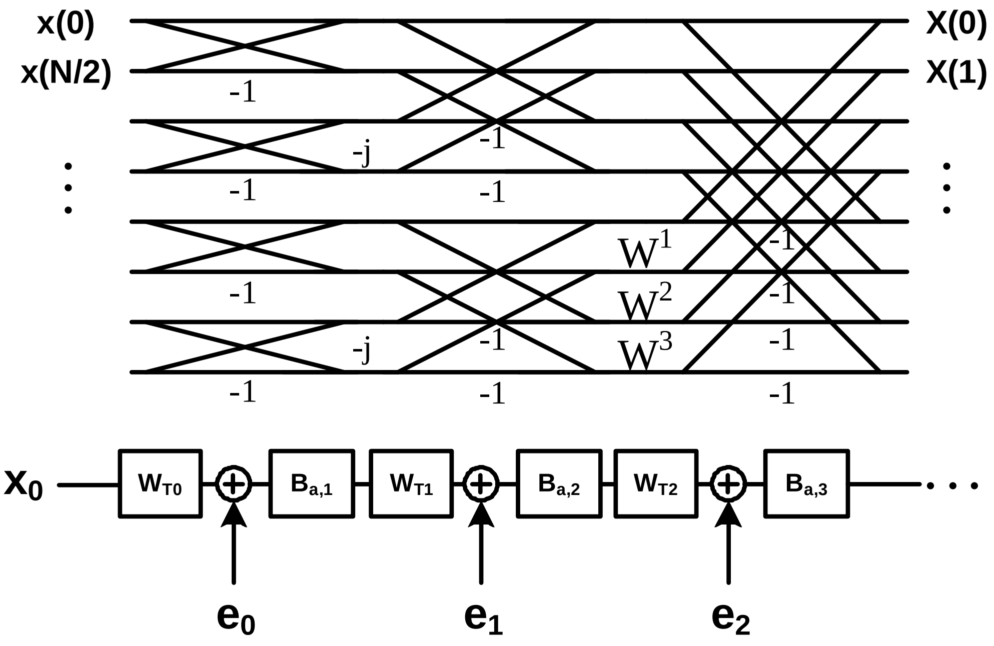


Figure 4.2: Propagation model of quantization loss for DIT FFT.

where w_{T_i} is the equivalent twiddle factor matrix at the ith stage of DIT FFT algorithms. Here we define n_{T_i} as the number of non-trivial twiddle factors at the ith stage. Also note that the signal flow of DIT algorithms is the reverse signal flow of DIF algorithms, we can also present the relationship between of w_{F_i} and w_{T_i} in (4.19).

$$w_{F_i} = w_{T_{\alpha-i-1}},\ 0 \leq i \leq \alpha - 1 \tag{4.19}$$

4.2 Noise Power Calculation

The subjective is to calculate the total noise power of the quantization loss. Therefore, we present the following lemmas which will be useful in later derivation.

Lemma 1: The matrix product of $B_{\alpha,i}$ and its own Hermitian matrix equals to $2I_{2^\alpha}$. Proof:

$$B_{\alpha,i}B_{\alpha,i}^H = (I_{2^{\alpha-i}} \otimes B_{i,i})(I_{2^{\alpha-i}} \otimes B_{i,i})^H$$

$$\begin{aligned}
&= (I_{2^{\alpha-i}} \otimes B_{i,i})(I_{2^{\alpha-i}}^H \otimes B_{i,i}^H) \\
&= I_{2^{\alpha-i}} \otimes (B_{i,i} B_{i,i}^H) \\
&= I_{2^{\alpha-i}} \otimes (H_2 \otimes I_{2^{i-1}})(H_2^H \otimes I_{2^{i-1}}^H) \\
&= I_{2^{\alpha-i}} \otimes (H_2 H_2^H) \otimes (I_{2^{i-1}} I_{2^{i-1}}^H) \\
&= 2(I_{2^{\alpha-i}} \otimes I_2 \otimes I_{2^{i-1}}) = 2I_{2^\alpha} \qquad (4.20)
\end{aligned}$$

Lemma 2: $w_{F_i} w_{F_i}^H = w_{F_i}^H w_{F_i} = I$ Proof:

$$\begin{aligned}
w_{F_i} w_{F_i}^H &= diag(w_{F_{i,j}} w_{F_{i,j}}^*) = I_{2^\alpha} \qquad (4.21) \\
& \quad 0 \le i \le \alpha - 1,\ 0 \le j \le 2^\alpha - 1
\end{aligned}$$

The total power of quantization noise of DIF FFT algorithms, P_{nf}, is calculated by $tr[E[\Delta X_F \Delta X_F^H]]$ as shown in (4.22). n_{F_i} is defined as the number of non-trivial complex multiplications at the ith stage of DIF FFT algorithms.

$$\begin{aligned}
P_{nf} &= tr[E[\Delta X_F \Delta X_F^H]] \\
&= tr[E[\sum_{i=0}^{\alpha-1} \prod_{j=i+1}^{\alpha-1} w_{F_j} B_{\alpha,\alpha-j} e_i \sum_{m=0}^{\alpha-1} e_m^H \prod_{n=m+1}^{\alpha-1} B_{\alpha,\alpha-n}^H w_{F_n}^H]] \\
&= tr[E[\sum_{i=0}^{\alpha-1} (\prod_{j=i+1}^{\alpha-1} w_{F_j} B_{\alpha,\alpha-j}) e_i e_i^H (\prod_{j=i+1}^{\alpha-1} B_{\alpha,\alpha-j}^H w_{F_j}^H)]] \\
&= E[\sum_{i=0}^{\alpha-1} tr[(\prod_{j=i+1}^{\alpha-1} w_{F_j} B_{\alpha,\alpha-j}) e_i e_i^H (\prod_{j=i+1}^{\alpha-1} B_{\alpha,\alpha-j}^H w_{F_j}^H)]] \\
&= E[\sum_{i=0}^{\alpha-1} tr[(e_i e_i^H)(\prod_{j=i+1}^{\alpha-1} B_{\alpha,\alpha-j}^H w_{F_j}^H)(\prod_{j=i+1}^{\alpha-1} w_{F_j} B_{\alpha,\alpha-j})] \\
&= E[\sum_{i=0}^{\alpha-1} tr[(e_i e_i^H)(\prod_{j=i+1}^{\alpha-1} B_{\alpha,\alpha-j}^H w_{F_j}^H w_{F_j} B_{\alpha,\alpha-j})]] \\
&= E[\sum_{i=0}^{\alpha-1} 2^{\alpha-i-1} tr[e_i e_i^H]] = \sum_{i=0}^{\alpha-1} 2^{\alpha-i-1} n_{F_i} \sigma_c^2 \qquad (4.22)
\end{aligned}$$

Again, we can calculate the overall noise power of DIT FFT algorithms as expressed in (4.23).

$$\begin{aligned}
P_{nt} &= tr[E[\Delta X_T \Delta X_T^H]] \\
&= tr[E[B_{\alpha,\alpha}\sum_{i=0}^{\alpha-1}(\prod_{j=i+1}^{\alpha-1} w_{T_j}B_{\alpha,j})e_i \sum_{m=0}^{\alpha-1} e_m^H(\prod_{n=m+1}^{\alpha-1} B_{\alpha,n}^H w_{T_n}^H)B_{\alpha,\alpha}^H]] \\
&= tr[E[B_{\alpha,\alpha}\sum_{i=0}^{\alpha-1}(\prod_{j=i+1}^{\alpha-1} w_{T_j}B_{\alpha,j})e_i e_i^H(\prod_{j=i+1}^{\alpha-1} B_{\alpha,n}^H w_{T_j}^H)B_{\alpha,\alpha}^H]] \\
&= E[tr[\sum_{i=0}^{\alpha-1}(\prod_{j=i+1}^{\alpha-1} w_{T_j}B_{\alpha,j})e_i e_i^H(\prod_{j=i+1}^{\alpha-1} B_{\alpha,n}^H w_{T_j}^H)B_{\alpha,\alpha}^H B_{\alpha,\alpha}]] \\
&= E[2\cdot\sum_{i=0}^{\alpha-1} tr[(e_i e_i^H)(\prod_{j=i+1}^{\alpha-1} B_{\alpha,n}^H w_{T_j}^H)(\prod_{j=i+1}^{\alpha-1} w_{T_j}B_{\alpha,j})] \\
&= E[2\cdot\sum_{i=0}^{\alpha-1} tr[(e_i e_i^H)(\prod_{j=i+1}^{\alpha-1} B_{\alpha,\alpha-j}^H w_{T_j}^H w_{T_j} B_{\alpha,\alpha-j})]] \\
&= E[\sum_{i=0}^{\alpha-1} 2^{\alpha-i} tr[e_i e_i^H]] = \sum_{i=0}^{\alpha-1} 2^{\alpha-i} n_{T_i}\sigma_c^2 \qquad (4.23)
\end{aligned}$$

From (4.22) and (4.23), we observe that the BF matrices, $B_{\alpha,i}$, acts as an internal amplifier which doubles the noise power after every stage. Therefore, if most of non-trivial twiddle factors are located in the later stages of FFT operation, the overall power of quantization loss will be smaller. In other words, generally speaking DIT FFT algorithms will be better than DIF algorithms since the twiddle factors of the former are mostly concentrated in the later stages, which agrees with the conclusion of [22]. Besides we can also compare the overall quantization noise power of different FFT algorithms by finding the number of non-trivial twiddle factors, n_{F_i} or n_{T_i}, at each stage. In order to make a quantitative comparison, without loss of generality, we assume all the quantization noises are mutually independent uniform random variables with variance σ_c^2, then the overall noise power of each FFT algorithms with alternative decimation schemes is listed in Table 4.1 and Table 4.2. Again, if the length of FFT operations is not power of 4, we use a cascaded radix-2 stage in the last stage for radix-4 DIF algorithms and a leading radix-2 stage for radix-4 DIT algorithms.

Table 4.1: Comparison of Overall Noise Power for DIF FFT (unit: σ_c^2)

FFT Size	Radix-2	Radix-4	Split-Radix
8	8	4	4
16	64	32	28
32	352	176	148
64	1664	832	684
128	7296	3648	2964
256	30720	15360	12396
512	126464	63232	50836
1024	514048	257024	206188

Table 4.2: Comparison of Overall Noise Power for DIT FFT (unit: σ_c^2)

FFT Size	Radix-2	Radix-4	Split-Radix
8	4	8	8
16	28	32	40
32	140	208	200
64	620	688	840
128	2604	3696	3528
256	10688	11760	14280
512	43180	60656	57800
1024	173740	191216	231880

4.3 Bit Allocation Optimization

According to (4.22) and (4.23), we can improve the SQNR performance by minimizing the quantization loss arised in the early stages of FFT operations if the wordlength of internal memory cells is not fixed for all stages. As for the implementation of FFT algorithms, most previous works address two design issues: the efficiency of complex multipliers and the memory strategy. The design parameters have to be adjusted according to different target applications. Designers focus on increasing the utilization rate of multipliers in the design of short-length FFT architecture. Conversely the memory access strategy becomes more important on designing a long-length FFT architecture. First we discuss the case of short-length FFT implementation. When considering the hardware design of short-length FFT architectures, delay feedback (DF) memory allocation is mostly used due to its higher utilization rate compared to that of the delay commutator (DC) scheme [12, 27]. The main idea of DF scheme is to store the first half of input samples in the DF memory cells and wait until the second half of input samples arrive. Although the total count of memory cells is the same for both types of FFT algorithms as expressed in (4.24) and (4.25), the number of memory cells of DIT algorithms will be doubled after every butterfly stage while that of the DIF algorithms will be half after each stage as shown in Fig. 4.3.

$$DIF: \ M_F = \sum_{i=0}^{\alpha-1} 2^{\alpha-i-1} = 2^\alpha - 1 \quad (4.24)$$

$$DIT: \ M_T = \sum_{i=0}^{\alpha-1} 2^{i} = 2^\alpha - 1 \quad (4.25)$$

Given the same input signals, we can try to minimize the quantization loss caused by internal arithmetic rounding operations to improve the overall SQNR performance by adjusting the wordlength of intermediate results. Largrange multiplier is a commonly adopted mathematical tool to find the local extreme of a multi-variable function subject to one or many constraint functions. Extra dummy variables, λ_i $(0 \leq i \leq k-1)$, are introduced to convert the original problem set with n variables and k constraint functions to a dual problem set with $n+k$ variables and no constraint function. Recall that the variance of complex multipliers,

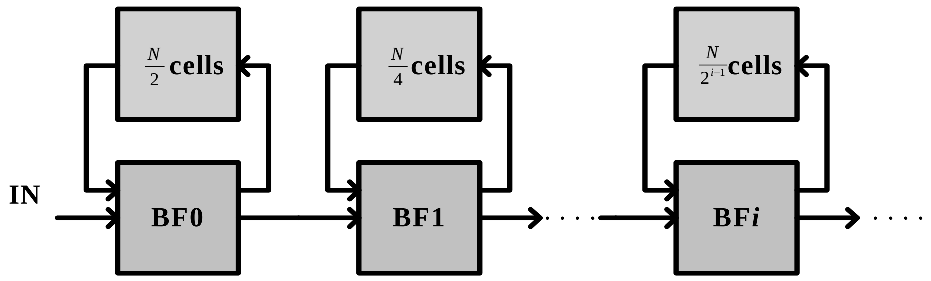


Figure 4.3: The block diagram of DF memory scheme for DIF FFT.

σ_c^2 can be represented by the MSE of real multiplications, σ^2. Without loss of generality, σ_c^2, σ^2 and internal bit length, b_c can be related in (4.26) for different implementations of complex multipliers, where $3 \leq \beta \leq 4$ [25].

$$\sigma_c^2 = \beta\sigma^2 = \beta\frac{2^{-2b_c}}{12} \tag{4.26}$$

For the DIT FFT implementations with DF memory scheme, the total number of memory cell, B_T, is expressed as:

$$B_T = 2 \cdot \sum_{i=0}^{\alpha-1} 2^i(b_i + b_x), \tag{4.27}$$

where b_x represents the effective wordlength of input samples and can be regarded as the minimal wordlength for internal memory storage. Besides, b_i is number of extra bits used in the ith stage of FFT computation.

To apply the Largrange multiplier optimization, the objective function is shown in (4.23) and the constraint function is the overall bit bidget expressed in (4.27). Therefore, the bit allocation can be represented as an optimization problem formulated in (4.28).

$$\begin{aligned} P_{nt}(b_i, \lambda_T) &= \sum_{i=0}^{\alpha-1} 2^{\alpha-i} n_{T_i} \beta \frac{2^{-2(b_i+b_x)}}{12} \\ &+ \lambda_T(2 \cdot \sum_{i=0}^{\alpha-1} 2^i(b_i + b_x) - B_T) \end{aligned} \tag{4.28}$$

By taking partial derivative on (4.28) with respect to b_i, we can calculate b_i in terms of λ_T as expressed in (4.30).

$$\frac{\partial P_{nt}}{\partial b_i} = 2^{\alpha-i} n_{T_i} \beta \frac{2^{-2(b_i+b_x)}}{12} \ln 2(-2) + 2^{i+1}\lambda_T \equiv 0 \tag{4.29}$$

$$b_i = \frac{1}{2}(\alpha - 2i - 1 + \log_2 n_{T_i} \beta \frac{\ln 2}{6} - \log_2 \lambda_T) - b_x \tag{4.30}$$

Also, by taking partial derivative on (4.28) with respect to λ_T, it yields:

$$\frac{\partial P_{nt}}{\partial \lambda_T} = 2 \cdot \sum_{i=0}^{\alpha-1} 2^i (b_i + b_x) - B_T \equiv 0 \tag{4.31}$$

Plug (4.30) into (4.31), we can calculate λ_T in (4.32) and (4.33). Then we plug (4.33) into (4.30) to find the expression of b_i as formulated in (4.34).

$$\begin{aligned}
\frac{\partial P_{nt}}{\partial \lambda_T} &= 2 \cdot \sum_{i=0}^{\alpha-1} 2^i (b_i + b_x) - B_T \\
&= (\alpha - 1 + \log_2 \beta \frac{\ln 2}{6} - \log_2 \lambda_T)(2^\alpha - 1) + \sum_{i=0}^{\alpha-1} 2^i \log_2 n_{T_i} \\
&\quad -(\alpha-2)2^{\alpha+1} - 4 - B_T \qquad (4.32) \\
\log_2 \lambda_T &= \frac{1}{2^\alpha - 1}[\sum_{i=0}^{\alpha-1} 2^i \log_2 n_{T_i} - (\alpha-2)2^{\alpha+1} - 4 - B_T] \\
&\quad +(\alpha - 1 + \log_2 \beta \frac{\ln 2}{6}) \qquad (4.33) \\
b_i &= \underbrace{\frac{1}{2(2^\alpha - 1)}((\alpha-2)2^{\alpha+1} + 4 + B_T - \sum_{i=0}^{\alpha-1} 2^i \log_2 n_{T_i}) - b_x}_{constants} \\
&\quad + \underbrace{\frac{1}{2}(\log_2 n_{T_i} - 2i)}_{variables} \qquad (4.34)
\end{aligned}$$

Similarly, for the DIF FFT implementations with DF memory scheme, the total number of bits used for memory cells, B_F, is expressed as:

$$B_F = 2 \cdot \sum_{i=0}^{\alpha-1} 2^{\alpha-i-1}(b_i + b_x) \tag{4.35}$$

In order to find the optimal bit allocation, we need to solve the Langrage equation expressed in (4.36), where the objective function is calculated in (4.22) and the constraint function is the number of overall bit budget as shown in (4.35).

$$\begin{aligned} P_{nf}(b_i, \lambda_F) &= \sum_{i=0}^{\alpha-1} 2^{\alpha-i-1} n_{F_i} \beta \frac{2^{-2(b_i+b_x)}}{12} \\ &+ \lambda_F(2 \cdot \sum_{i=0}^{\alpha-1} 2^{\alpha-i-1} b_i - B_F) \end{aligned} \tag{4.36}$$

First of all, by taking partial derivative on (4.36) with respect to b_i, we can express b_i in term of λ_F as:

$$\frac{\partial P_{nf}}{\partial b_i} = 2^{\alpha-i-1} n_{F_i} \beta \frac{2^{-2(b_i+b_x)}}{12} \ln 2(-2) + 2^{\alpha-i} \lambda_F \equiv 0 \tag{4.37}$$

$$b_i = \frac{1}{2}(\log_2 n_{F_i} \beta \frac{\ln 2}{6} - 1 - \log_2 \lambda_F) - b_x \tag{4.38}$$

Secondly, we can take the partial derivative on (4.36) with respect to λ_F as:

$$\frac{\partial P_{nf}}{\partial \lambda_F} = 2 \cdot \sum_{i=0}^{\alpha-1} 2^{\alpha-i-1}(b_i + b_x) - B_F \equiv 0 \tag{4.39}$$

Plug (4.38) into (4.39), we can calculate λ_F in (4.37) and (4.41). Therefore, by inserting (4.41) into (4.38), we can find the expression of b_i for DIF FFT algorithms as formulated in (4.42).

$$\begin{aligned} \frac{\partial P_{nf}}{\partial \lambda_F} &= \sum_{i=0}^{\alpha-1} 2^{\alpha-i-1}(b_i + b_x) - B_F \\ &= (\log_2 \beta \frac{\ln 2}{6} - 1 - \log_2 \lambda_F)(2^\alpha - 1) + \sum_{i=0}^{\alpha-1} 2^{\alpha-i-1} \log_2 n_{F_i} \\ &\quad - B_F & (4.40) \\ \log_2 \lambda_F &= (\log_2 \beta \frac{\ln 2}{6}) - 1 + \frac{1}{(2^\alpha - 1)}(\sum_{i=0}^{\alpha-1} 2^{\alpha-i-1} \log_2 n_{F_i} - B_F) & (4.41) \\ b_i &= \frac{1}{2}[\underbrace{\log_2 n_{F_i}}_{variables} + \underbrace{\frac{1}{2^\alpha - 1}(B_F - \sum_{i=0}^{\alpha-1} 2^{\alpha-i-1} \log_2 n_{F_i})] - b_x}_{constants} & (4.42) \end{aligned}$$

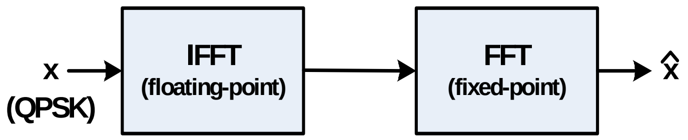


Figure 4.4: Fixed-point simulation with QPSK signals.

4.4 Quantitative Comparison Results

4.4.1 Fixed-Point Simulation of Different FFT Algorithms

In order to verify the expressions derived in the previous section, a fixed-point simulation environment is implemented as shown in Fig. 4.4. First, floating point inverse fast Fourier transform (IFFT) is performed with random QPSK signals. The wordlength of twiddle factors coefficients is also set to 10 bits. The internal wordlength of fixed-point FFT is swept from 8 bits to 18 bits. Both 512-point and 1024-point FFTs are simulated. As seen from Fig. 4.5, the SQNR value of all 1024-point simulations is lower than that of 512-point simulations because the internal arithmetic rounding errors will accumulate as the length of FFT increases. Based on observation of Fig. 4.6, radix-2 DIT algorithms have the best accuracy of all. For both radix-2 and radix-4 FFT algorithms, the DIT versions have better SQNR performance than the DIF versions, which agrees with the general rules described in the previous section. However, if split-radix FFT algorithm is adopted, the DIF version will outperform its own DIT version , which is as expected from Table 4.1 and Table 4.2. It is also clear to see from Fig. 4.6 that all decompositions except radix-2 DIF FFT show comparative performance.

4.4.2 Bit Allocation of Short-Length FFT Architecture

In order to compare the accuracy of fixed-point FFTs with different internal wordlength configurations, three different settings of internal wordlength are specified. The simulation flow is described as follows. First of all, a 128-point floating-point IFFT is performed with QPSK signals. The transformed output of IFFT computation is fed into an 8-bit quantized then passed to fixed-point DIT

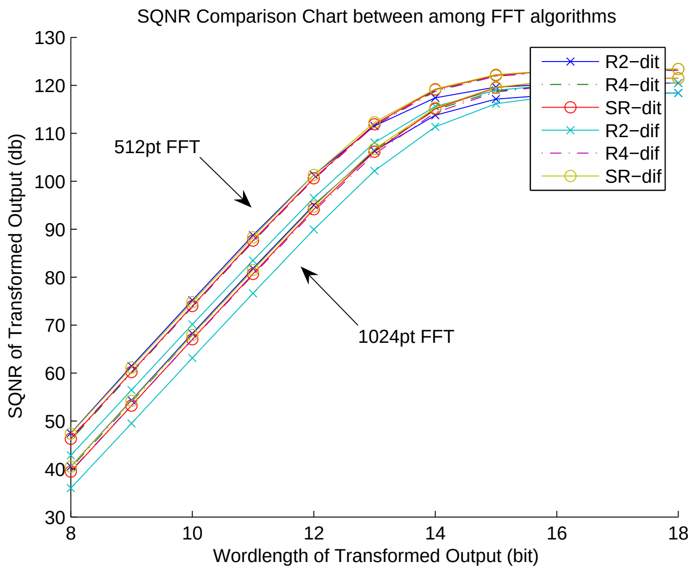


Figure 4.5: SQNR comparison chart among FFT algorithms with fixed-point arithmetic.

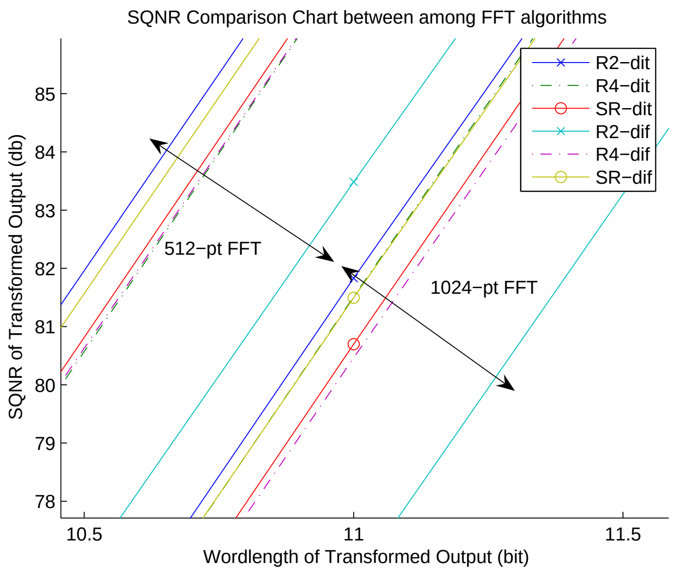


Figure 4.6: Zoom-in version of SQNR comparison chart among FFT algorithms with fixed-point arithmetic.

Table 4.3: Wordlength setting of each intermediate memory stages.

	Wordlength (bits)						
	1st	2nd	3rd	4th	5th	6th	7th
Case (Inc)	8	9	10	11	12	13	14
Case (Same)	11	11	11	11	11	11	11
Case (Opt)	14	15	15	15	12	12	9

Table 4.4: Comparison of Different Memory Scheme, 128-pt DIT SR-FFT

	SQNR (dB)	Change(%)	Bit Count	Change(%)
Case (Inc)	73.1031	0.00%	3316	0.00%
Case (Same)	73.4597	0.49%	2794	-15.74%
Case (Opt)	73.4853	0.52%	2752	-17.01%

split-radix FFT to calculate the SQNR performance. The number of the first row, Case (Inc), listed in Table 4.3 shows the results from increasing 1 bit after every BF stage. The second row, Case (Same), shows that all intermediate results are stored in 11-bit memory cells. The third row shows the memory scheme recommended from the optimization equation derived in last section. 1000 trials are made. As observed from Table 4.4, blindly increasing the internal wordlength by 1 bit will use most memory usage and lead to worst performance compared to the other two configurations. For the comparison between the second case and the third case, it is obvious that the optimized wordlength setting will yield comparative simulation results with fewer bit counts.

Similarly, the simulation is also made for 256-pt DIF FFT. The internal wordlength is set as shown in Table 4.5. An 8-bit quantizer is used to limit the dynamic range of transformed IFFT output. The simulation results are shown in Table 4.6.

Table 4.5: Wordlength of each intermediate memory stages.

	Wordlength (bits)							
	1st	2nd	3rd	4th	5th	6th	7th	8th
Case (Inc)	8	9	10	11	12	13	14	15
Case (Same)	11	11	11	11	11	11	11	11
Case (Opt)	8	11	11	11	11	11	11	11

Table 4.6: Comparison of different memory schemes, 256-pt DIF split-radix FFT.

	SQNR (dB)	Change(%)	Bits	Change(%)
Case (Inc)	65.9116	0.00%	4574	0.00%
Case (Same)	66.4539	0.82%	5610	22.65%
Case (Opt)	66.4539	0.82%	4842	5.86%

4.4.3 Simulation with the Presence of Noisy Channel

The main bottleneck of DF-based architecture is the number of concurrent memory access increases as the length of FFT increases. This leads to the requirement of multiple separated memory blocks and duplicate BF modules. Therefore, for the design of long-length FFT architecture, both the accuracy of transformed output and the feasible memory access scheme become important issues. Several previous researches have addressed these two design obstacles. Choi et al. [32] proposed CBFP method to dynamically normalize the intermediate results after each stages. In [33], the authors also utilize the idea of CBFP with an on-the-fly normalization module so as to remove the usage of temporary buffers. In [29] and [30], the authors increased the efficiency of memory access by inserting two caches and reported that the memory blocks are the largest blocks for long-length FFT architecture. Generally speaking, the designs of long-length FFT architecture will use centralised memory blocks to save hardware cost. As a result, we can not freely adjust the wordlength of intermediate memory storages at every

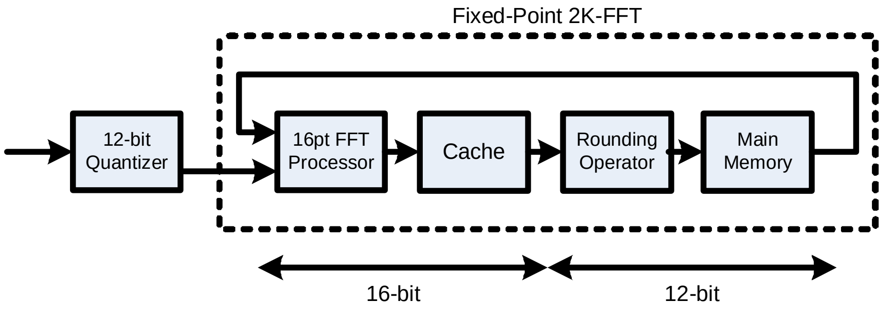


Figure 4.7: The block diagram of 2048-point fixed-point FFT simulation.

stage. Recall from (4.22) and (4.23), we can reduce the overall noise power by assigning more bits in the early stage of FFT computation. Therefore, in order to achieve better accuracy, a 2048-point split-radix DIF FFT architecture with improved FFT processor is simulated. As shown in Fig. 4.7, the wordlength of main memory block is 12 bits and that of FFT core and internal cache is set to 16 bits. Scaling operations are performed after every butterfly stage to maintain the dynamic range. A simulation environment with frequency selective channel is built to observe the effect of bit error rate (BER) with improved FFT processor. The channel is modeled as a finite length FIR filter as a quasi-static model. It has 3 consecutive paths which are located at integer chip position. Their relative power profile is [0 0 0] dB, which is power normalized so that the overall average channel impulse response energy is equal to 1. The noise at the receiver is assumed to be additive white complex Gaussian noise. The signal-to-noise ratio (SNR) is defined as ratio of the average signal tone power to the noise power at a frequency bin. In order to estimate the average BER, at least 1000 channel realizations are simulated for each SNR point The receiver is assumed to have perfect knowledge of the channel state information. The signal flow is shown in Fig. 4.8. The channel SNR is swept from 0 dB to 40 dB. As we can see from Fig. 4.9, the long-length FFT computation with improved FFT processor can achieve better BER under high-SNR conditions.

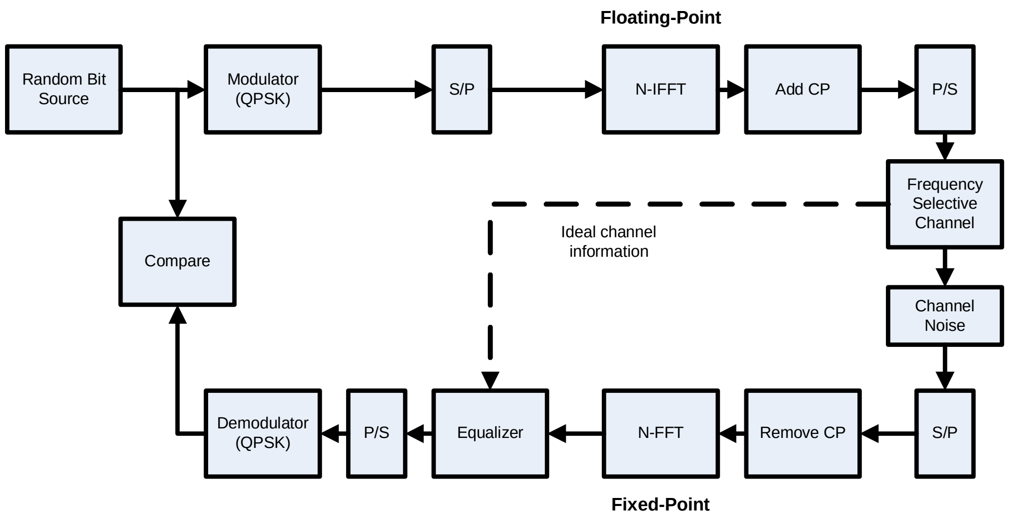


Figure 4.8: The block diagram of 2048-point fixed-point FFT simulation with frequency selective fading channel.

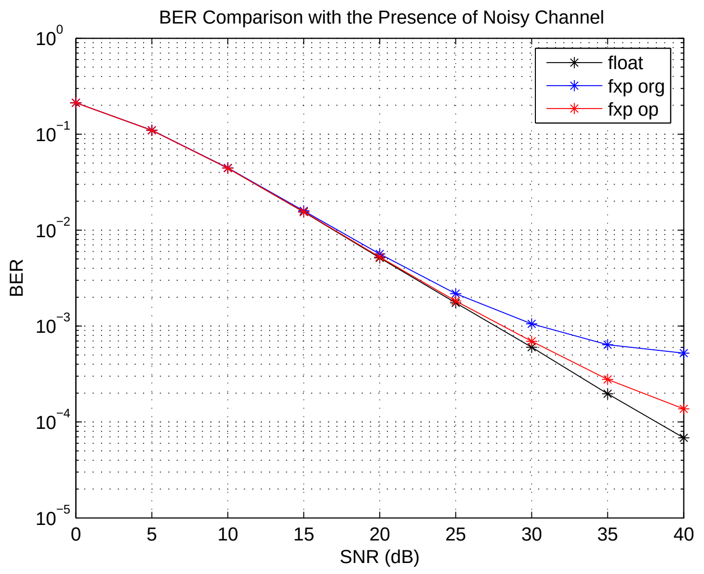


Figure 4.9: The bit error rate comparison with improved FFT processor.

5

Parallel Integer FFT Architecture Design for MB-OFDM

In this chapter, a parallel VLSI architecture based on mixed-radix IntFFT for the upcoming MB-OFDM [7–9] system is proposed. The periodicity property of lifting coefficients proposed in Chapter 3 and the concurrent relationship of non-trivial multiplications are both utilized to reduce the hardware cost of complex multipliers. The proposed mixed-radix FFT architecture is based on Eq. (2.58) and Eq. (2.59) to arrange the complex multiplications. The I/O behavior of the proposed architecture is in a parallel fashion with four concurrent samples. The proposed design uses DF memory strategy to keep the intermediate results. Four data paths are required for each sub-modules. The periodicity of lifting coefficients, the multiple constant multiplier (MCM) technique and multiplication reordering are utilized to reduce the complexity of multipliers and to increase the hardware utilization.

5.1 System Requirement of MB-OFDM System

The rapidly increasing demand of OFDM-based applications, including wireless LAN [6] and MB-OFDM systems [7–9] makes processing speed a major consideration in FFT architecture design. As such the study of high-performance VLSI FFT architecture is of increasing importance.

The structure of the MB-OFDM system is similar to that of a conventional wireless LAN physical layer. The main difference is that the carrier frequency is varied with the time-frequency code. The MB-OFDM system is designed for low-complexity solutions by limiting the transmitted signals with only QPSK constellation to reduce the usage of internal memory in the digital baseband processing. The system adopts an OFDM scheme with 128 sub-carriers [9], which leads to the requirement of a 128-pt FFT/IFFT architecture. The bandwidth of the transmitted signals is 528Mhz and the symbol length is 312.5ns for all data rates. A guard interval (9.5 ns) is appended and a prefix interval (60.6 ns) is inserted for each OFDM symbol. In other words, the desirable FFT/IFFT architecture has to complete one OFDM symbol within the information interval (242.2 ns).

5.2 Previous Work on FFT Architecture

Numerous researches have been done for FFT architectures in the past decades. As for the implementation of FFT algorithms, most previous works address two design issues: the efficiency of complex multipliers and the memory strategy. The design parameters have to be adjusted according to different target applications. Designers focus on increasing the utilization rate of multipliers in the design of short-length FFT architecture. In [26], the authors proposed a coefficient reordering scheme with the commutator architecture to reduce the switching activity. Yeh and Jen [27] proposed a data rescheduling scheme by bit-reverse and bit-inverse indexing to reduce the number of required multipliers. Han et al. [28] achieve the same purpose by sharing the subexpressions between multiplicands to improve the utilization rate of complex multipliers. Conversely the memory access strategy becomes more important on designing a long-length FFT architecture. In [29] and [30], the authors increase the memory access efficiency by inserting two caches between the functional units and the main memory banks. Kuo et al. [31] also utilize the memory-cache architecture and expand it with the ability of variable-length FFT computations. Because the proposed architecture is targeted at the MB-OFDM system, the main design challenge is how to reduce the complexity of

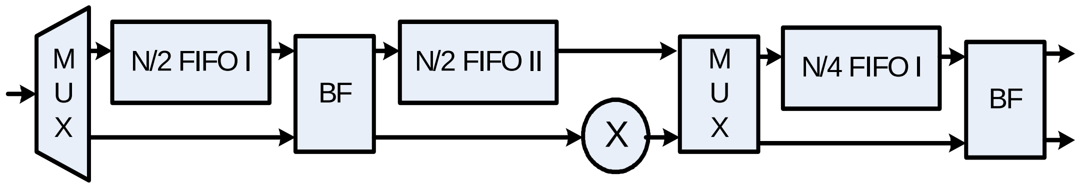


Figure 5.1: Delay commutator buffering strategy.

multipliers.

5.3 Memory Access Strategy

Take a N-point radix-2 DIF algorithm as example, the computation cannot start until both $x(n)$ and $x(n+\frac{N}{2})$ are available. For sequential input, the two samples will be separated by $\frac{N}{2}$ cycles if only one sample is input per clock cycle. As a result, the first half input samples have to be stored in internal memory storages until the second half samples arrive. There are mainly two different approaches: DC [47] as shown in Figure 5.1 and DF as shown in Figure 5.2 [12]. The main idea of DC memory scheme is to balance the odd- and even-frequency part of FFT computation by inserting internal buffers. During the first $\frac{N}{2}$ cycles, the first half samples are stored in "$\frac{N}{2}$ first-in-first-out (FIFO) I". At the next $\frac{N}{2}$ cycles, the butterfly receives $x(n)$ from "$\frac{N}{2}$ FIFO I" and $x(n+\frac{N}{2})$ from input ports. Meanwhile, the butterfly send one output into "$\frac{N}{2}$ FIFO II" and the other one output into complex multipliers. During the N cycles,data are stored into the "$\frac{N}{2}$ FIFO I" in the first $\frac{N}{2}$ cycles and then are read from the FIFO in the second $\frac{N}{2}$ cycles. Hence, the utilization rate of each FIFO is only 50%.

For the DF memory scheme, in the first half cycles, the BF_i core will simply store the input samples into the feedback memory. After the first $N/2$ cycles, the BF_i core retrieves the $x(n)$ samples from the feedback memory, performs corresponding operations with the sample $x(n+N/2)$ and then feeds the output into the next BF_{i+1} core. The necessary number of memory cells for the k^{th} stage is $\frac{N}{2^k}$. Therefore, by mathematical induction, the total required memory is $N-1$ cells.

In the proposed architecture, a multiple delay feedback (MDF) scheme as illus-

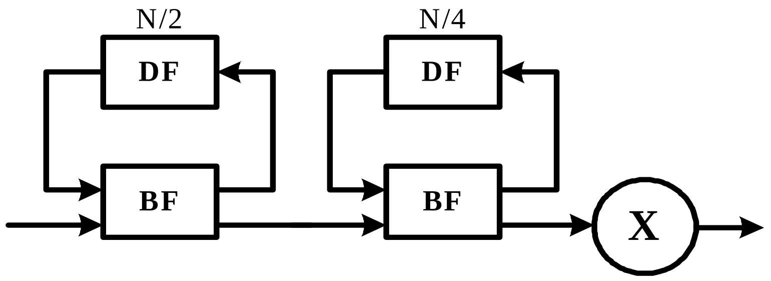


Figure 5.2: Delay feedback buffering strategy.

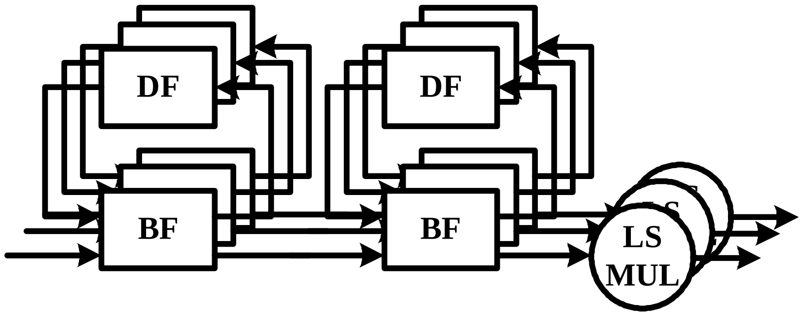


Figure 5.3: Multiple delay feedback buffering strategy.

trated in Figure 5.3 to process four input samples in every cycle. The basic idea of MDF is similar to that of DF so that the overall number of internal memory cells remains the same. However, multiple BF modules are required to deal with multiple input samples concurrently. In order to reduce the hardware cost of complex multiplers, the concurrency of input data is analyzed in the next section.

5.4 Concurrency Analysis of Complex Multiplications

According to Eq. (2.59), the radix-4 FFT shows better spatial regularity since only trivial multiplications are used for the first quarter input samples. The concurrent lifting coefficient groups used in the first and the second multiplier stage are shown in Table 5.1 and Table 5.2 respectively, where U_i indicate the *ith* pair of the universal coefficient set. Two constant multipliers are necessary in the first

Table 5.1: Scheduling of the constant multipliers in the first multiplier stage.

Time Slot	0	1
1st MUL	0	0
2nd MUL	0	U_{16}
3rd MUL	0	0
4th MUL	$-i$	$-U_{16}$

Table 5.2: Scheduling of the constant multipliers in the second multiplier stage.

Time Slot	0/1	2	3	4	5	6	7
1st MUL	0	0	0	0	U_{16}	U_{12}	U_4
2nd MUL	0	U_8	$-U_8$	U_4	U_{12}	U_8	U_8
3rd MUL	0	U_{16}	$-U_{16}$	U_8	U_8	$-U_4$	U_{12}
4th MUL	0	U_8	$-U_8$	U_{12}	U_4	$-U_{16}$	0

multiplier stage. For the second multiplier stage, only 3 constant multipliers and one MCM are required after rescheduling the order of complex multiplications as listed in Table 5.3.

Since simultaneously only four twiddle factors are used due to the parallel input fashion, certain coefficients may never appear in the same cycle. By applying the selective criteria of optimal coefficient set described in Chapter 3, totally sixteen possible combinations of coefficient groups may appear in the third stage. Observed from the concurrency analysis of concurrent multiplicands shown in Fig. 5.4, six mutual exclusive subsets of the universal set for the third stage are found as: $\{1, 6, 13\}$, $\{2, 12, 16\}$, $\{3, 7, 8\}$, $\{4, 9, 14\}$, $\{5, 10, 15\}$, and $\{11\}$. This observation leads to hardware saving by applying the MCM technique [41,48]. As illustrated in Fig. 5.5 and Fig. 5.6, the lifting coefficients are first represented in the canonical signed digit (CSD) format [49, 50] to reduce the number of additions. The digits of the same group with the dark grey background form a saving group; same as

Table 5.3: Reordering of the complex multiplications in the second multiplier stage.

Time Slot	0/1	2	3	4	5	6	7
1st MUL	0	0	0	U_{12}	U_{12}	U_{12}	U_{12}
2nd MUL	0	U_8	$-U_8$	U_8	U_8	U_8	U_8
3rd MUL	0	U_8	$-U_8$	U_4	$-U_4$	$-U_4$	U_4
4th MUL	0	U_{16}	$-U_{16}$	0	$-U_{16}$	$-U_{16}$	0

the digits with light grey background. For example, by defining partial products, $p_0 = (101)_{CSD}$ and $p_1 = (10\bar{1})_{CSD}$, each coefficients in the second subset of P coefficients can be rewritten as Eq. (5.1)-(5.3) with the partial products. Similarly, by defining partial product $q_0 = (\bar{1}01)_{CSD}$ and $q_1 = (\bar{1}00\bar{1})_{CSD}$, the second subset of Q coefficients can also be represented as Eq. (5.4)-(5.6) to share the common adders. For 12-bit lifting coefficients, the total number of adders is reduced from 131 to 78, which is a 40.4% saving. Routing network and multiplexers are necessary to arrange the input samples along with corresponding MCMs. An additional 4-bit control signal is also required to indicate the desired coefficient combination. The hardware diagram of MCM is shown in Fig. 5.7. In brief, no general multiplier is used to perform the lifting operations in the proposed design. Totally five MCMs and eight constant multipliers (two for the first stage, five for the second stage and the other one for the third stage) are necessary.

$$U_{P_2} = (000010\bar{1}00101) = p_1 \ll 5 + p_0 \tag{5.1}$$

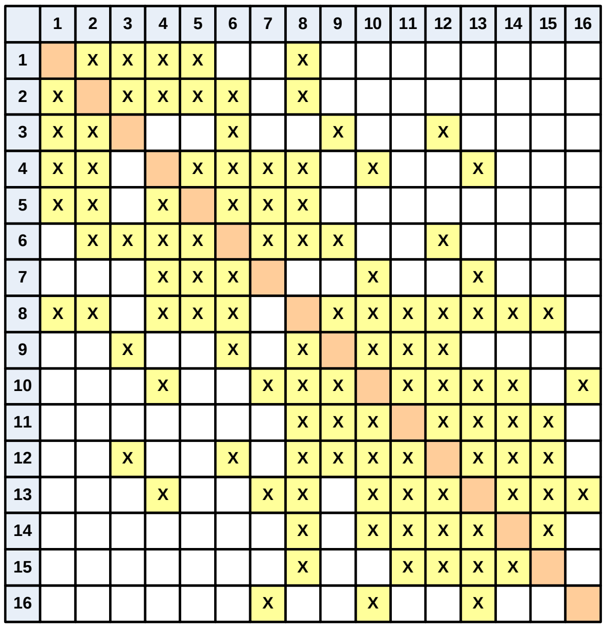
$$U_{P_{12}} = (0010100\bar{1}0\bar{1}01) = p_0 \ll 7 - p0 \ll 2 + 1 \tag{5.2}$$

$$U_{P_{16}} = (010\bar{1}01010000) = p_1 \ll 8 + p_0 \ll 4 \tag{5.3}$$

$$U_{Q_2} = (000\bar{1}0100\bar{1}0\bar{1}) = q_0 \ll 6 + q_1 \tag{5.4}$$

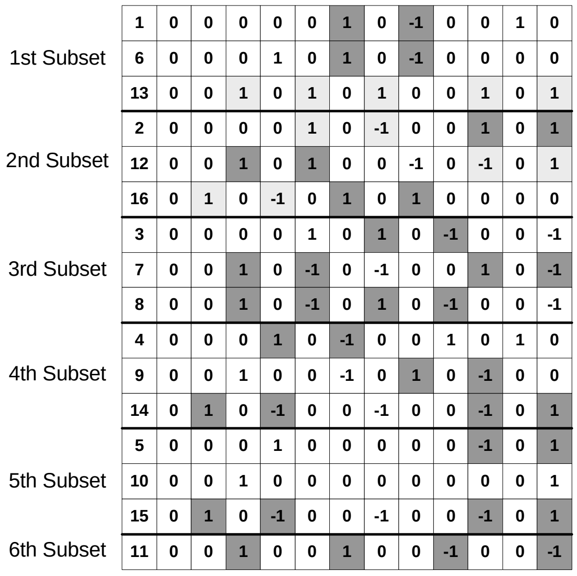
$$U_{Q_{12}} = (0\bar{1}00\bar{1}0010000) = q_1 \ll 6 + 10000 \tag{5.5}$$

$$U_{Q_{16}} = (\bar{1}01010\bar{1}0\bar{1}000) = q0 \ll 9 - q0 \ll 5 - 1000 \tag{5.6}$$



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		X	X	X	X			X								
2	X		X	X	X	X		X								
3	X	X				X			X			X				
4	X	X			X	X	X	X		X			X			
5	X	X		X		X	X	X								
6		X	X	X	X		X	X	X			X				
7				X	X	X				X			X			
8	X	X		X	X	X			X	X	X	X	X	X	X	
9			X			X		X		X	X	X				
10				X			X	X	X		X	X	X	X		X
11								X	X	X		X	X	X	X	
12			X			X		X	X	X	X		X	X	X	
13				X			X	X		X	X	X		X	X	X
14								X		X	X	X	X		X	
15								X			X	X	X	X		
16							X			X			X			

Figure 5.4: Concurrency analysis of the 3rd stage multiplier.



	1	0	0	0	0	0	1	0	-1	0	0	1	0
1st Subset	6	0	0	0	1	0	1	0	-1	0	0	0	0
	13	0	0	1	0	1	0	1	0	0	1	0	1
	2	0	0	0	0	1	0	-1	0	0	1	0	1
2nd Subset	12	0	0	1	0	1	0	0	-1	0	-1	0	1
	16	0	1	0	-1	0	1	0	1	0	0	0	0
	3	0	0	0	0	1	0	1	0	-1	0	0	-1
3rd Subset	7	0	0	1	0	-1	0	-1	0	0	1	0	-1
	8	0	0	1	0	-1	0	1	0	-1	0	0	-1
	4	0	0	0	1	0	-1	0	0	1	0	1	0
4th Subset	9	0	0	1	0	0	-1	0	1	0	-1	0	0
	14	0	1	0	-1	0	0	-1	0	0	-1	0	1
	5	0	0	0	1	0	0	0	0	0	-1	0	1
5th Subset	10	0	0	1	0	0	0	0	0	0	0	0	1
	15	0	1	0	-1	0	0	-1	0	0	-1	0	1
6th Subset	11	0	0	1	0	0	1	0	0	-1	0	0	-1

P_i Coefficient in CSD Format

Figure 5.5: MCM representation of $P[n]$ of the first radix-4 FFT stage.

	1	0	0	0	0	-1	0	1	0	0	-1	0	0
1st Subset	6	0	0	-1	0	0	-1	0	-1	0	-1	0	1
	13	0	-1	0	-1	0	1	0	0	0	-1	0	0
	2	0	0	0	-1	0	1	0	0	-1	0	0	-1
2nd Subset	12	0	-1	0	0	-1	0	0	1	0	0	0	0
	16	-1	0	1	0	1	0	-1	0	-1	0	0	0
	3	0	0	0	-1	0	-1	0	1	0	1	0	-1
3rd Subset	7	0	-1	0	1	0	1	0	1	0	0	0	0
	8	0	-1	0	1	0	0	0	-1	0	0	0	0
	4	0	0	-1	0	1	0	0	-1	0	0	0	0
4th Subset	9	0	-1	0	0	1	0	0	1	0	1	0	0
	14	0	-1	0	-1	0	0	0	-1	0	-1	0	1
	5	0	0	-1	0	0	0	0	1	0	0	-1	0
5th Subset	10	0	-1	0	0	0	1	0	0	0	-1	0	-1
	15	-1	0	1	0	1	0	1	0	0	0	0	1
6th Subset	11	0	-1	0	0	0	0	-1	0	0	1	0	-1

Q_i Coefficient in CSD Format

Figure 5.6: MCM representation of $Q[n]$ of the first radix-4 FFT stage.

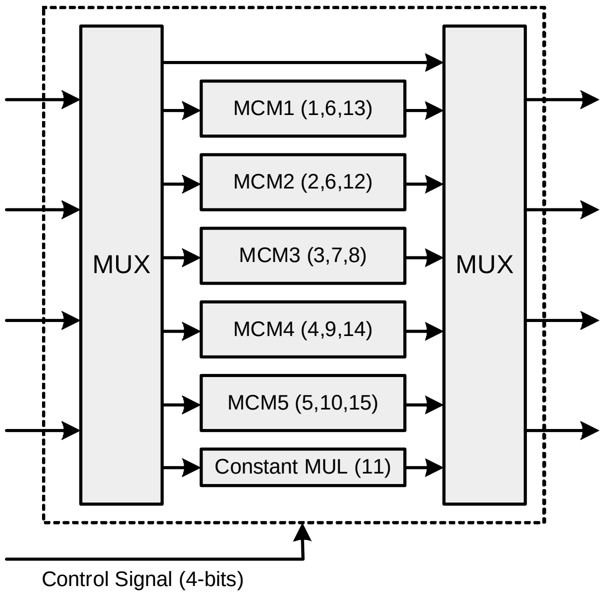


Figure 5.7: The hardware block diagram of proposed MCM.

Table 5.4: Control signal of different decompositions.

Control Signal (C)	00	01	10	11
Decomposition Type	Type (a)	Type (b)	Type (c)	Type (c)

Table 5.5: Intermediate variables of each LS-MAC stage.

Variable	Description
R_0	$x_r + \rho \oplus (Px_i) + \rho U_{ip} + \rho$
I_0	$\rho \oplus x_i$
R_1	R_0
I_1	$I_0 + s \oplus (R_0 U_{iq}) + s + c[1]$

5.5 The Digital Arithmetic of Lifting Operations

In general, the lifting operations can be regards as a cascaded two-port network, where one multiplication and one addition are involved in each stage. In order to accommodate four alternative decompositions and preserve the SCI property, a specialized multiply-and-add accumulator (MAC) for lifting operations is proposed. Assume two input x_r and x_i as shown in Fig. 5.8, a two bit control signal, C, listed in Table 5.4 is used to indicate the current decomposition type and s represents forward operation ($s = 0$) or inverse operation ($s = 1$). The addition is combined as the last stage of the Wallace tree of the multipliers. Besides, in order to save the additional adder used in the sign inverse of two's complement numbers, the increment adder is also merged into later operations. As the result, if we define $\rho = s \oplus c[1]$, the temporary variable after each MAC stage can be characterized in Table 5.5, where U_{ip} and U_{iq} are the P and Q coefficients of the ith pair of the universal coefficient set. Therefore, by merging the addition and increment into multiplication, each lifting operations can be performed by one constant MAC. Similarly for the output of Type (b) and Type (d), the increment of sign inverse is propagated to next BF stage as an additional carry-in input.

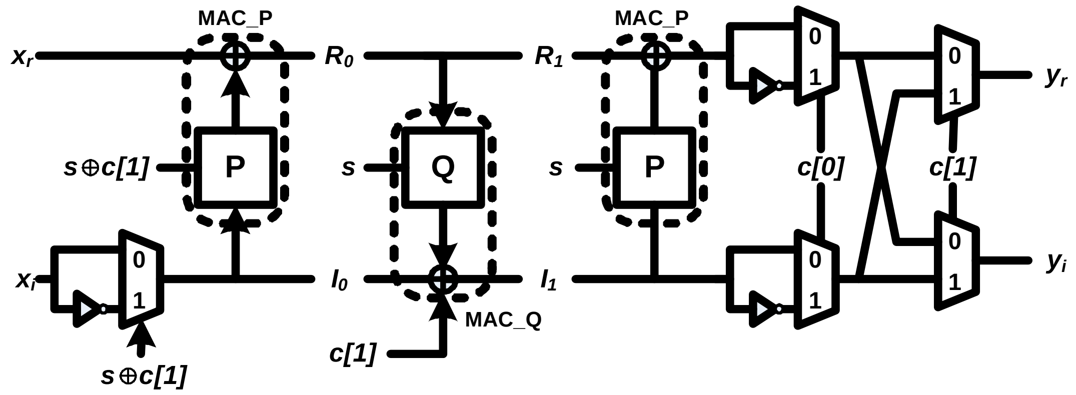


Figure 5.8: The general LS architecture with specialized MAC.

5.6 Hardware Resource Comparison

The proposed architecture utilized MDF scheme to store temporary results. Four samples are input at every cycle. The initial pipeline delay is 32 cycles. It takes 32 cycles to complete one 128-point FFT/IFFT operation. The hardware resource comparison to previous designs is highlighted in Table 5.6. As we can see, the MDF-based architecture uses less memory usage than multiple delay commutator (MDC) designs as presented in [47] and [51]. Compare to [44], the proposed design has higher efficiency of complex multipliers because no general multiplier is required.

5.7 System-Level BER Performance

A Simulink-based UWB physical layer model [54] depicted in Fig. 5.9 is used to verify the performance of IntFFT. Both IntFFT and FxpFFT are simulated. The transmission rate of the simulation platform is targeted at 200Mb/s, which is the highest mandatory rate of MB-OFDM standard. The result of floating point FFT is also illustrated as the upper bound of system performance. In order to reduce the usage of internal memory and maintain lower performance degradation, the simulation is performed with different wordlength. As seen in Fig. 5.10, the minimal acceptable wordlength of proposed architecture is 12 bits, which agrees

Table 5.6: Hardware resource comparison of the 128-pt parallel FFT architectures.

	R2MDC [47]	Garcia [51]	Zhang [52]	Lin [53]	Proposed
Registers	190	190	544	124	124
Complex Multipliers	6	10	3	2 + 4x0.62	5 MCMs + 6 CMs
Complex Adders	14	34	48	48	48
FFT Algorithm	Radix-2	Split-Radix	Radix-2^2 Radix-2	Radix-8 Radix-2	Radix-2^2 Radix-2
Input Fashion	Parallel (2 ports)	Parallel (6 ports)	Parallel (4 ports)	Parallel (4 ports)	Parallel (4 ports)
Output Fashion	Parallel (2 ports)	Parallel (6 ports)	Parallel (4 ports)	Parallel (4 ports)	Parallel (4 ports)
Multiplier	General	General	CORDIC	General	LS
Memory	MDC	MDC	Loopback	MDF	MDF

with the conclusions from [55]. The system-level performance comparison made between IntFFT and FxpFFT is shown in Fig. 5.11. As listed in Table 5.7, IntFFT could achieve better BER performance compared to FxpFFT in most cases.

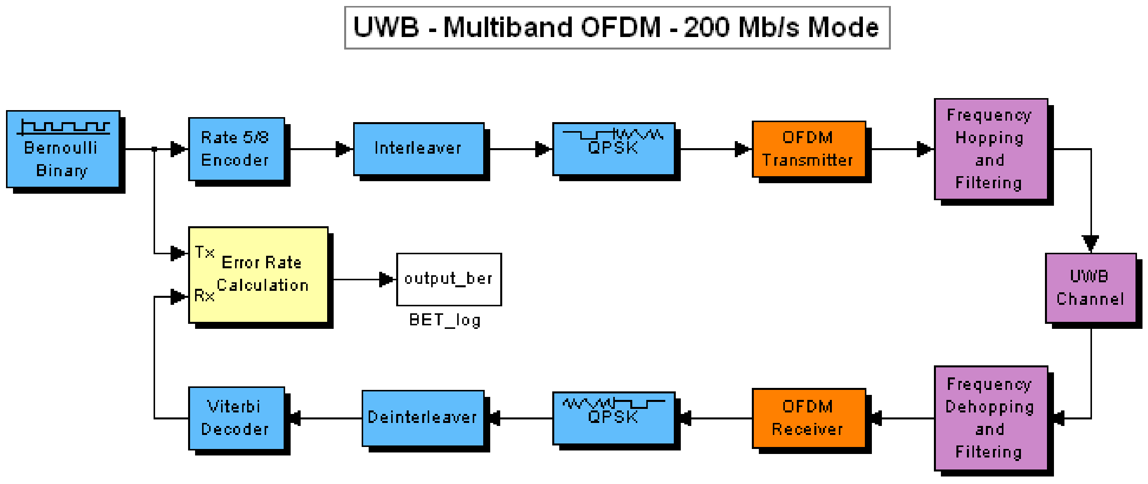


Figure 5.9: Simulink-based simulation platform of UWB physical layer.

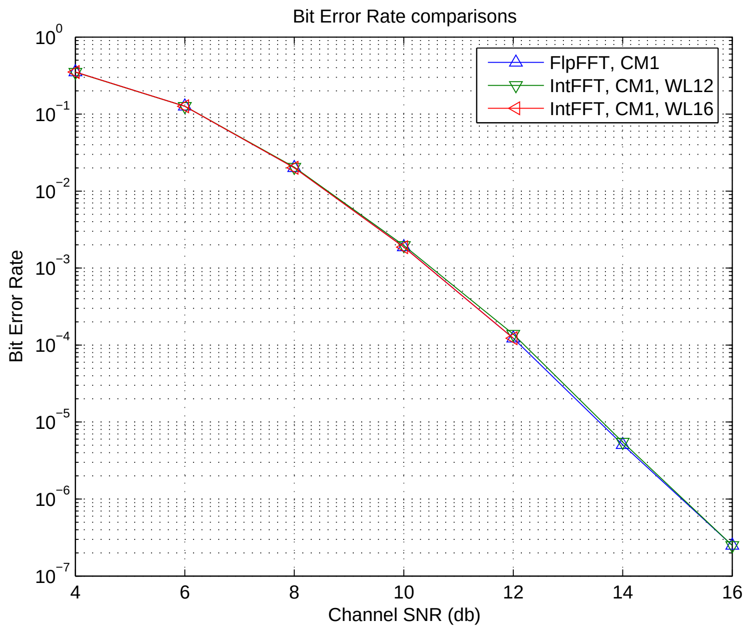


Figure 5.10: BER comparison with different internal wordlength.

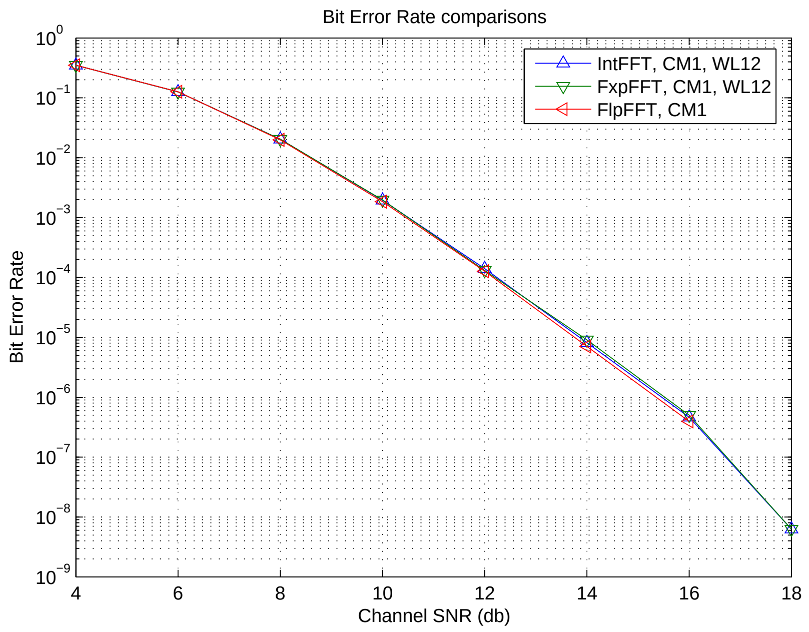


Figure 5.11: BER comparison between IntFFT and FxpFFT.

Table 5.7: BER improvement ratio of Integer FFT.

Channel SNR	FxpFFT	IntFFT	Improvement (%)
4	3.4953×10^{-1}	3.4965×10^{-1}	-0.0347
6	1.2601×10^{-1}	1.2600×10^{-1}	0.0045
8	2.0316×10^{-2}	2.0306×10^{-2}	0.0486
10	1.9714×10^{-3}	1.9501×10^{-3}	1.0789
12	1.3103×10^{-4}	1.4120×10^{-4}	-7.7599
14	9.0694×10^{-6}	8.1381×10^{-6}	10.2688
16	5.0003×10^{-7}	4.6253×10^{-7}	7.5000

6

Pipeline Integer FFT Architecture Design for 802.11a

In this chapter, a pipeline VLSI architecture based on radix-2^2 IntFFT is proposed to demonstrate its efficiency. The IntFFT algorithm guarantees the PR property of transformed samples. For a 64-points radix-2^2 FFT architecture, the proposed architecture uses 2 sets of complex multipliers (6 real multipliers) and has 6 pipeline stages. The whole design is synthesized and simulated with a 0.18μm TSMC 1P6M standard cell library and its reported equivalent gate count usage is only 17,963 gates. Each module of the proposed architecture is introduced in the following section.

6.1 Butterfly Core

Two signals are used to control the BF core as shown in Figure 6.1: BF_i_BP and BF_i_CTRL. The former is a one bit control signal and is asserted to 1 during the first half of the DF operations, which means that there is no actual operation and the input sample will be fed into the DF memory bank directly. The BF_i_CTRL control signal for each BF module is a 2-bit variable. Assuming that I_1 and I_2 are the input samples to the BF core, I_{1r} and I_{2r} respectively denote the real parts, and I_{1i} and I_{2i} stand for the imagine part. In Table 6.1, the functionality of four different BF_i_CTRL modes is listed. Mode "00" and "11" are used when $\pm j$ is

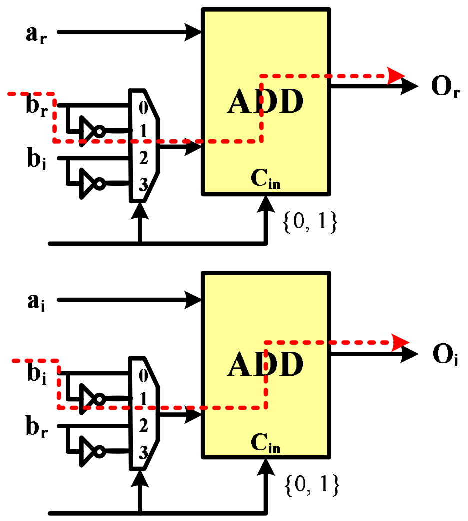


Figure 6.1: Illustration of BF core.

Table 6.1: List of BF_i_CTRL operation modes.

Mode	Function	Description
00	$I_1 + I_2$	$O_r = I_{1r} + I_{2r}$; $O_i = I_{1i} + I_{2r}$
11	$I_1 - I_2$	$O_r = I_{1r} + \sim I_{2r} + 1$; $O_i = I_{1i} + \sim I_{2i} + 1$
01	$I_1 + jI_2$	$O_r = I_{1r} + \sim I_{2i} + 1$; $O_i = I_{1i} + I_{2r}$
10	$I_1 - jI_2$	$O_r = I_{1r} + I_{2r}$; $O_i = I_{1i} + \sim I_{2r} + 1$

not present. The remaining two modes, "01" and "10", are issued when a $\pm j$ term occurs. In order to save the adders used in two's complement conversion, the additional +1 addition is merged into latter adders as a carry-in input as is shown in Figure 6.1. Because the critical path of the FFT architecture is determined by the multipliers, this method saves on hardware cost and does not degrade the overall performance.

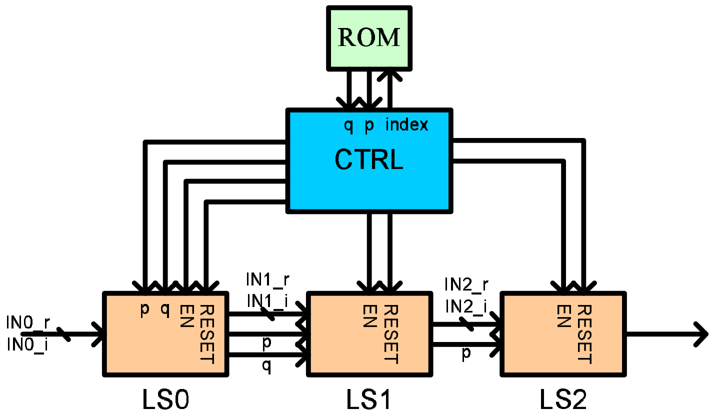


Figure 6.2: Architecture view of coefficient passing scheme.

6.2 Coefficient Passing Scheme

Observing from the lifting decomposition of the IntFFT algorithm in Figure 2.9, it is seen that the first and third lifting coefficients are identical. This property makes it possible to share lookup table (LUT) values for the lifting coefficients. A coefficient passing scheme is proposed here to achieve this idea. The LUTs of coefficients are no longer constructed with individual lifting blocks but instead with the global CTRL module. The coefficients are generated corresponding to the current status of lifting stages and passed with the input samples to cascaded lifting stages. For example, using the first set of lifting multipliers between BF_1 and BF_2, CTRL module will only pass P_0 and Q_0 to the LS0 module. Within the LS0 module, two register cells are required for temporary storage of lifting coefficients. In the next cycle, P_0 and Q_0 will be passed to the cascaded stages along with the corresponding input data. Concurrently, the CTRL module will send new values of P_1 and Q_1 to LS0 for computation of the next cycle. Note that in the third lifting stage, only the P_i coefficient is necessary, therefore there is only one set of register cells required for the LS1 module. In summary, three additional register sets are used to save one 32-to-1 MUX and one P_i coefficient LUT. The architectural view of the proposed design and its timing behavior diagram are shown in Figure 6.2 and Figure 6.3 respectively.

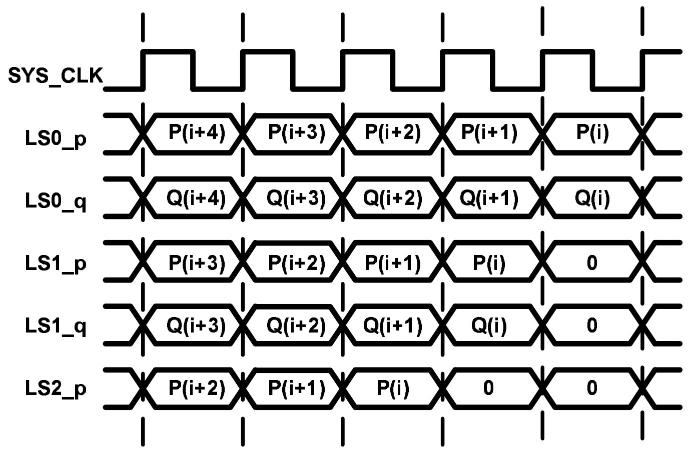


Figure 6.3: Timing behavior of coefficient passing scheme.

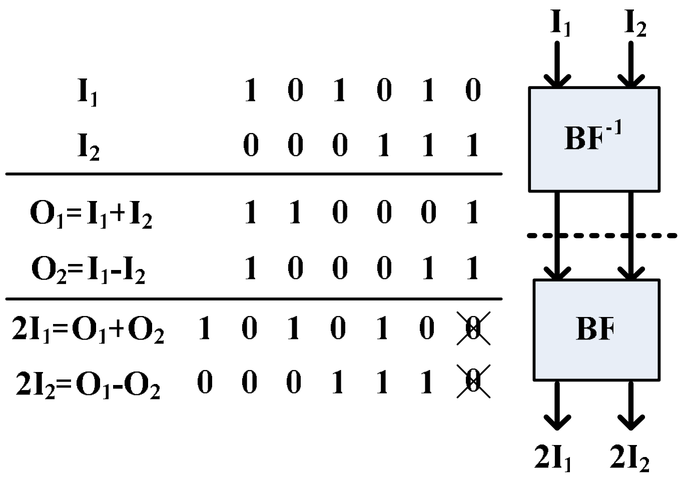

I_1 1 0 1 0 1 0
I_2 0 0 0 1 1 1
$O_1=I_1+I_2$ 1 1 0 0 0 1
$O_2=I_1-I_2$ 1 0 0 0 1 1
$2I_1=O_1+O_2$ 1 0 1 0 1 0 $\cancel{0}$
$2I_2=O_1-O_2$ 0 0 0 1 1 1 $\cancel{0}$
I_1
I_2
BF^{-1}
BF
$2I_1$
$2I_2$

Figure 6.4: Example of balanced BF operations.

6.3 Memory Strategy

As mentioned in Chapter 5, the memory strategy of the BF stage is an important issue for FFT architecture design. There are mainly two different approaches: DC [47] and DF [12] introduced in previous chapter respectively. In our architecture, the latter is adopted and the projection view of the overall architecture design is depicted in Figure 6.5.

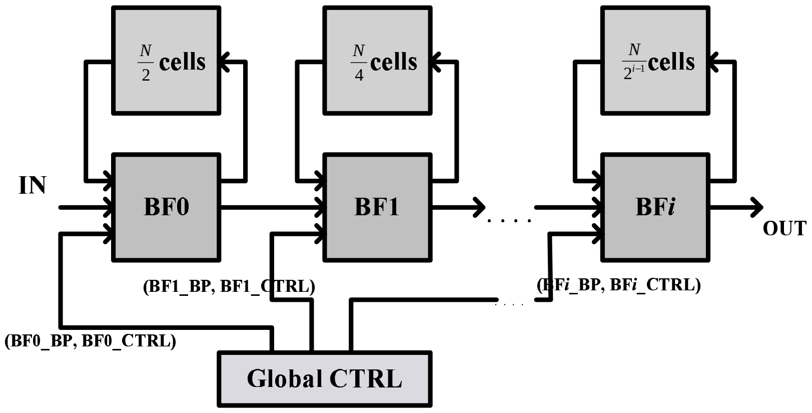


Figure 6.5: Projection view of DF memory scheme.

6.4 SQNR Comparison

In order to verify the accuracy of the IntFFT against the conventional FxpFFT, both algorithms are implemented in Matlab for comparison. The system test flow is as follows. First, a function library of binary operators is constructed as the fundamental components. Second, the Matlab version IntFFT is implemented to verify that the PR property is preserved and that the output agrees with the Verilog version design. Third, the FxpFFT is implemented and the SQNRs of the transformed outputs of both algorithms are calculated. $10,000$ independent trials were run to obtain the average SQNR, as plotted in Figure 6.6. Based on the simulation results, when the wordlength for the lifting coefficients and the twiddle factors is set to 12 bits, both the IntFFT and the FxpFFT yield sufficient SQNR performance when the internal wordlength is 12 bits.

6.5 BER Comparison in 802.11a Wireless LAN Systems

An OFDM-based wireless LAN system as shown in Figure 6.7 is constructed in Simulink for system level simulation. A multipath frequency-selective fading channel is used to evaluate the BER versus different channel SNR values. The wordlength of the coefficients is set to 12 bits. Various modulation schemes includ-

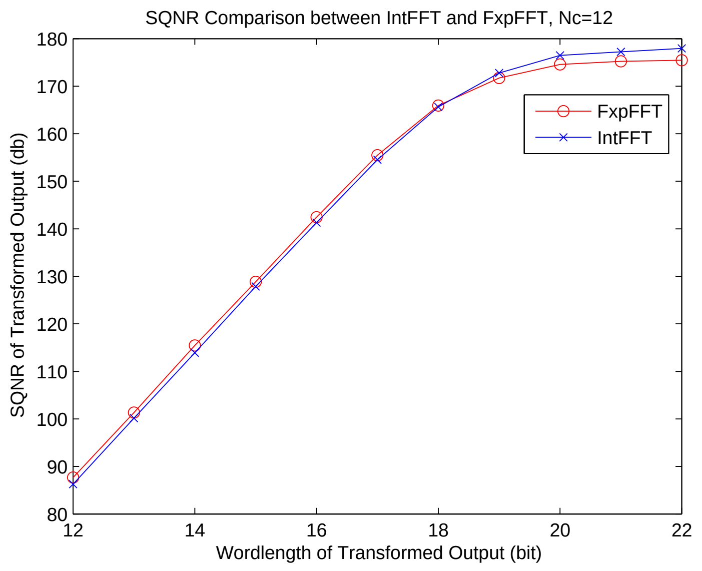


Figure 6.6: The SQNR comparison between IntFFT and FxpFFT, Nc = 12.

ing QPSK (1/2), 16-quadrature amplitude modulation (QAM) (1/2) and 64-QAM (1/2) are simulated. From Figure 6.8, it is seen that IntFFT performs as well as conventional FxpFFT even if the noisy channel is present.

6.6 Design Statistics

The proposed design is implemented in Verilog HDL and simulated by Verilog-XL. After the RTL level, it is synthesized by Buildgates and the AP&R flow is implemented by SOC Encounter with a TSMC 0.18-μm 1P6M process standard cell library. After the AP&R process, the generated standard delay format (SDF) file is back annotated to the Verilog-XL simulator in order to verify if the proposed design is correct. Then, the power consumption analysis is performed by VoltageStorm. In the design, the larger delay feedback memory blocks are implemented with pre-generated memory hard macros. The input sequence uses 12-bit for both real and imaginary parts. The maximum system clock is specified at 200-Mhz during the synthesis stage. The final design is a pad limited design. The core size is 500μm x 500μm, with a core utilization of 80%. The whole chip size is 975μm x 977μm,

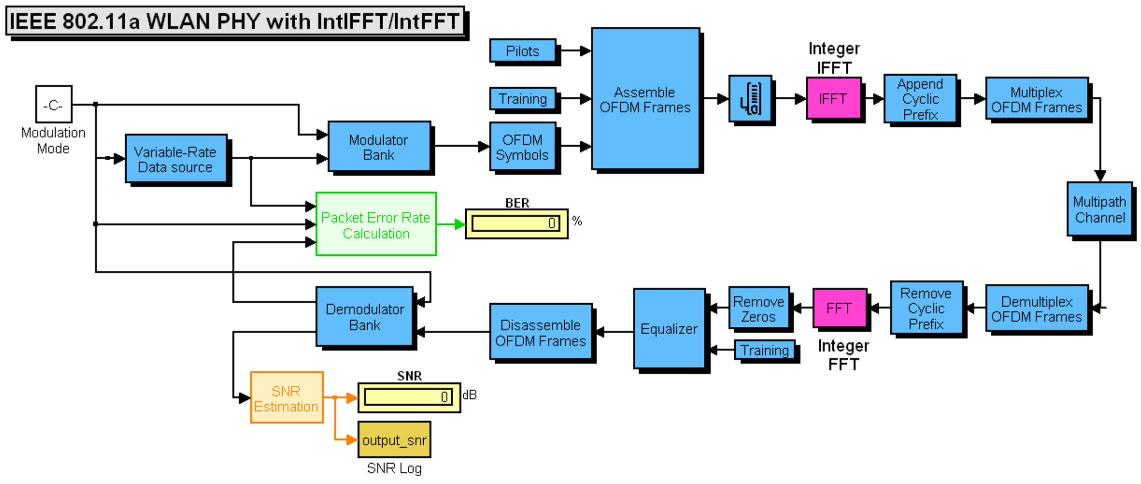


Figure 6.7: Schematic of 802.11a Simulink simulation platform for IntFFT.

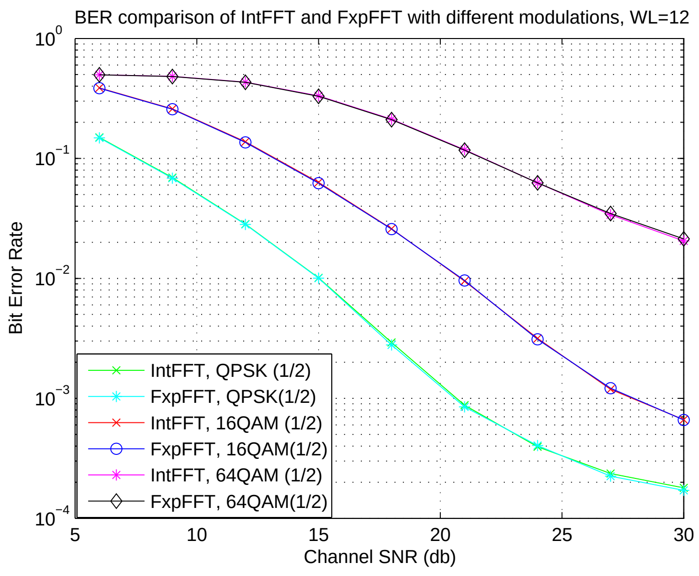


Figure 6.8: BER performance of IntFFT in 802.11a wireless LAN system.

Table 6.2: Comparisons of different FFT architecture

Design	Architecture	Gate Count	Critical Path	Power Consumption
[27]	Split-Radix	38,168	6.09 ns	507.85 mW (100Mhz, 3.3V)
Proposed	Radix-2^2	17,963	5.00ns	83.56 mW (200Mhz, 1.8V)

with 39 data pins, 8 power pins and 1 filler pin. The reported equivalent gate count is $17,983$ gates. The estimated core power consumption is 83.56-mW. The gate count usage for each of the major components is listed in Table 6.3. The layout view is shown in Figure 6.9.

6.7 Hardware Resource Comparisons

The hardware resource comparisons of several classical FFT architecture designs are highlighted in Table 6.4. From this comparison of different architectures, we can conclude that:

1) All the DF-based designs require smaller memory size than DC-based designs [47].

2) In Comparing radix-2^2 based designs [12] to radix-2 design [56], the former uses fewer number of complex multipliers and can access the memory banks more efficiently.

3) Comparing the radix-2^2 based design to the split-radix based design, the hardware cost of both is the same in terms of the number of adders and required memory size, while the multiplicative complexity of the latter is less.

Table 6.3: Area usage of each building blocks

Category	CTRL	BF Core	MUL_LS	Feedback Memory
Area	8.68%	25.67%	28.63%	36.31%

Table 6.4: Hardware resource usage comparisons

Architecture	Complex Multipliers	Complex Adders	Multiplicative
R2MDC [47]	$2(\log_4 N - 1)$	$4log_4N$	Radix-2
R2SDF [56]	$2(\log_4 N - 1)$	$4log_4N$	Radix-2
R4SDF [57]	$\log_4 N - 1$	$8log_4N$	Radix-4
R2^2SDF [12]	$\log_4 N - 1$	$4log_4N$	Radix-4
SRSDF [27]	$\log_4 N - 1$	$4log_4N$	Split-Radix
Proposed[1]	$\log_4 N - 1$	$4log_4N$	Radix-4

4) The proposed design requires less memory usage than previous radix-2^2 based design [12].

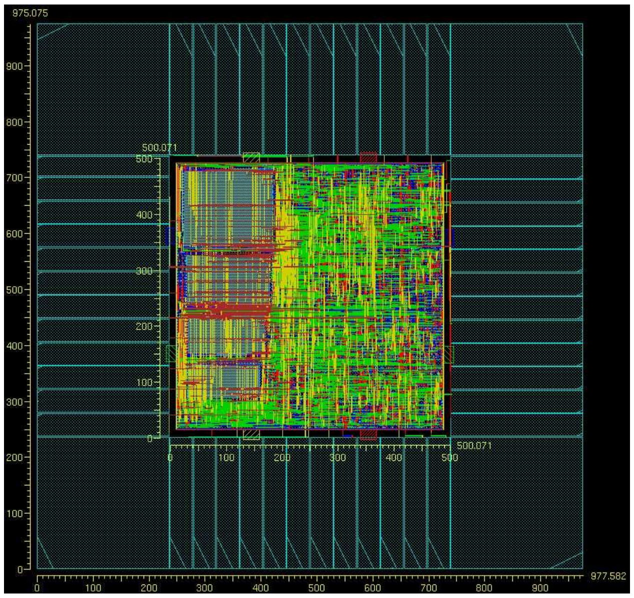


Figure 6.9: Layout view of proposed FFT architecture design.

7

Conclusion and Future Work

To conclude, we summarize the work on fixed-point analysis of FFT algorithms and the architecture design of IntFFT for two wireless communication standards.

7.1 Conclusion

In this thesis, the IntFFT with optimized coefficient set is proposed and quantitively compared to other popular FFT implementations. The advantage of using the selected coefficient set are twofold: First, it utilizes the periodicity of lifting coefficients so as to reduce the size of coefficient read-only memory (ROM) since only $\frac{N}{8}$ coefficients are enough to represent all twiddle factors for length-N FFT computation. Besides, it also leads to better accuracy because the MSE with the optimized coefficient set is lower. Based on the quantitive simulation results of Figure 3.6 and Figure 3.7, IntFFT with optimized coefficient set has better SQNR performance compared to other implementations.

Secondly, a comprehensive study is presented for the effect of fixed-point arithmetic in FFT operations. We have derived a general analytic expressions for the noise power of overall arithmetic rounding errors for all FFT algorithms. The theoretical derivations show the significance of the location of the non-trivial complex multiplications. Because the quantization noise power will be amplified after each butterfly stage, the less introduced noise power in the early stage will lead to better performance. Compared to previous works, a general propagation model is proposed to quickly estimate the arithmetic quantization errors of different FFT algorithms. From the simulation results most FFT algorithms yield similar performance except radix-2 DIF FFT. An operational optimization procedure is also proposed to reduce the hardware cost without sacrificing SQNR performance for short-length FFT architecture. From the simulation results, the FFT modules with higher accuracy will lead to better BER with the presence of noisy channel.

Lastly, a pipeline architecture for 802.11a wireless LAN and a parallel architecure for MB-OFDM are proposed. The proposed architectures focus on improving the efficiency of complex multipliers for non-trivial twiddle factors. The former utilizes the PR property to reduce the memory usage and yields comparative BER performance. By coefficient reordering and finding the concurrent relationship of non-trivial multiplications, the common terms of different constant multipliers within the same mutual exclusive set are shared to reduce the hardware cost used for complex multipliers in the proposed parallel architecture. Furthermore, a specialized LS-MAC architecture is also proposed for different LS decomposition types and can perform both forward and inverse lifting operations with a simple control logic.

7.2 Future Work

Many problems still have not been solved completely. Below, we outline some works and research directions that can be continued in the future.

7.2.1 Variable-length FFT Architecture Design

With the latest development of broadband wireless communication [58–60], the long-length FFT with ability to deal with variable-length input vectors is getting important. Most of the existent long-length FFT architectures target at a particular length and utilize the regularity to reduce the multiplicative complexity [33, 61, 62]. However, with the new consideration of variable-length requirement, a feasible architecture should meet the throughput requirement without sacrificing design generality. Thus, a dynamic cache-based FFT processor to handle variable-length input vectors is increasingly desired.

7.2.2 Fixed-point Analysis of FFT Algorithms with BFP Operations

In the long-length FFT architecture, block floating point (BFP) arithmetic is used to dynamically adjust the exponent of intermediate results to increase the dynamic range and improve the overall SQNR performance. Further research could be done to study the effect of fixed-point arithmetic for long-length FFT computation as both scaling operation and BFP arithmetic are used.

Bibliography

[1] S. Kumar, M. Biswas, and T. Nguyen, "Efficient phase correlation motion estimation using approximate normalization," in *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on*, vol. 2, 2004, pp. 1727–1730 Vol.2.

[2] U. Reimers, "Digital video broadcasting," *Communications Magazine, IEEE*, vol. 36, no. 6, pp. 104–110, 1998.

[3] ——, "DVB-the family of international standards for digital video broadcasting," *Proceedings of the IEEE*, vol. 94, no. 1, pp. 173–182, 2006.

[4] G. Faria, J. Henriksson, E. Stare, and P. Talmola, "DVB-H: digital broadcast services to handheld devices," *Proceedings of the IEEE*, vol. 94, no. 1, pp. 194–209, 2006.

[5] U. Ladebusch and C. Liss, "Terrestrial DVB (dvb-t): a broadcast technology for stationary portable and mobile use," *Proceedings of the IEEE*, vol. 94, no. 1, pp. 183–193, 2006.

[6] "Wireless LAN medium access control (MAC) and physical layer (PHY) specification," IEEE, Piscataway, NJ, 1997.

[7] "Multi-band OFDM physical layer proposal for IEEE 802.15 task group 3a," 03268r3P802-15-TG3a-Multi-band-CFP-Document.doc, 2003. [Online]. Available: http://grouper.ieee.org/groups/802/15/pub/2003/Jul03

[8] J. Balakrishnan, A. Batra, and A. Dabak, "A multi-band OFDM system for UWB communication," in *Ultra Wideband Systems and Technologies, 2003 IEEE Conference on*, 2003, pp. 354–358.

[9] A. Batra, J. Balakrishnan, G. Aiello, J. Foerster, and A. Dabak, "Design of a multiband OFDM system for realistic UWB channel environments," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 52, no. 9, pp. 2123–2138, 2004.

[10] J. W. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, apr 1965.

[11] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, 2nd ed. Pretice Hall, 1998.

[12] S. He and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)modulation," in *Signals, Systems, and Electronics, 1998. ISSSE 98. 1998 URSI International Symposium on*, 1998, pp. 257–262.

[13] H. Sorensen, M. Heideman, and C. Burrus, "On computing the split-radix FFT," *Acoustics, Speech, and Signal Processing, IEEE Transactions on*, vol. 34, no. 1, pp. 152–156, 1986.

[14] L. Jia, Y. Gao, and H. Tenhunen, "Efficient VLSI implementation of radix-8 FFT algorithm," in *Communications, Computers and Signal Processing, 1999 IEEE Pacific Rim Conference on*, 1999, pp. 468–471.

[15] S. Bouguezel, M. Ahmad, and M. Swamy, "Improved radix-4 and radix-8 FFT algorithms," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, vol. 3, 2004, pp. III–561–4 Vol.3.

[16] S. Bouguezel, M. O. Ahmad, and M. Swamy, "An alternate approach for developing higher radix FFT algorithms," in *Circuits and Systems, 2006. APCCAS 2006. IEEE Asia Pacific Conference on*, 2006, pp. 227–230.

[17] A. Oppenheim and C. Weinstein, "Effects of finite register length in digital filtering and the fast fourier transform," *Proceedings of the IEEE*, vol. 60, no. 8, pp. 957–976, 1972.

[18] T. Thong and B. Liu, "Fixed-point fast Fourier transform error analysis," *Acoustics, Speech, and Signal Processing, IEEE Transactions on*, vol. 24, no. 6, pp. 563–573, 1976.

[19] R. Meyer, "Error analysis and comparison of FFT implementation structures," in *Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 International Conference on*, 1989, pp. 888–891 vol.2.

[20] Y. Ma, "An accurate error analysis model for fast Fourier transform," *Signal Processing, IEEE Transactions on*, vol. 45, no. 6, pp. 1641–1645, 1997.

[21] D. James, "Quantization errors in the fast Fourier transform," *Acoustics, Speech, and Signal Processing, IEEE Transactions on*, vol. 23, no. 3, pp. 277–283, 1975.

[22] R. Perlow and T. Denk, "Finite wordlength design for VLSI FFT processors," in *Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on*, vol. 2, 2001, pp. 1227–1231 vol.2.

[23] D. Chandra, "Accumulation of coefficient roundoff error in fast Fourier transforms implemented with logarithmic number system," *Acoustics, Speech, and Signal Processing, IEEE Transactions on*, vol. 35, no. 11, pp. 1633–1636, 1987.

[24] S. Y. Park and N. I. Cho, "Fixed-point error analysis of CORDIC processor based on the variance propagation formula," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 51, no. 3, pp. 573–584, 2004.

[25] W.-H. Chang and T. Nguyen, "Integer FFT with optimized coefficient sets," in *To be appeared on Acoustics, Speech and Signal Processing, 2007. ICASSP 2007 Proceedings. 2006 IEEE International Conference on*, 2007.

[26] M. Hasan, T. Arslan, and J. Thompson, "A novel coefficient ordering based low power pipelined radix-4 FFT processor for wireless LAN applications," *Consumer Electronics, IEEE Transactions on*, vol. 49, no. 1, pp. 128–134, 2003.

[27] W.-C. Yeh and C.-W. Jen, "High-speed and low-power split-radix FFT," *Signal Processing, IEEE Transactions on*, vol. 51, no. 3, pp. 864–874, 2003.

[28] W. Han, T. Arslan, A. Erdogan, and M. Hasan, "Novel low power pipelined FFT based on subexpression sharing for wireless LAN applications," in *Signal Processing Systems, 2004. SIPS 2004. IEEE Workshop on*, 2004, pp. 83–88.

[29] B. Baas, "A low-power, high-performance, 1024-point FFT processor," *Solid-State Circuits, IEEE Journal of*, vol. 34, no. 3, pp. 380–387, 1999.

[30] Y.-W. Lin, H.-Y. Liu, and C.-Y. Lee, "A dynamic scaling FFT processor for DVB-T applications," *Solid-State Circuits, IEEE Journal of*, vol. 39, no. 11, pp. 2005–2013, 2004.

[31] J.-C. Kuo, C.-H. Wen, C.-H. Lin, and A.-Y. Wu, "VLSI design of a variable-length FFT/IFFT processor for ODFM-based communication systems," *EURASIP Journal on Applied Signal Processing*, vol. 13, pp. 1306–1316, 2003.

[32] J.-R. Choi, S.-B. Park, D.-S. Han, and S.-H. Park, "A 2048 complex point FFT architecture for digital audio broadcasting system," in *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, vol. 5, 2000, pp. 693–696 vol.5.

[33] T. Lenart and V. Owall, "A 2048 complex point FFT processor using a novel data scaling approach," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, vol. 4, 2003, pp. IV–45–IV–48 vol.4.

[34] M. Frigo and S. Johnson, “The design and implementation of FFTW3,” *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[35] A. Gupta and V. Kumar, “The scalability of FFT on parallel computers,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 4, no. 8, pp. 922–932, 1993.

[36] V. Boriakoff, “FFT computation with systolic arrays, a new architecture,” *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on]*, vol. 41, no. 4, pp. 278–284, 1994.

[37] Y. T. Ma, “A VLSI-oriented parallel FFT algorithm,” *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on]*, vol. 44, no. 2, pp. 445–448, 1996.

[38] F. Franchetti, S. Kral, J. Lorenz, and C. Ueberhuber, “Efficient utilization of SIMD extensions,” *Proceedings of the IEEE*, vol. 93, no. 2, pp. 409–425, 2005.

[39] G. Bi and E. Jones, “A pipelined FFT processor for word-sequential data,” *Acoustics, Speech, and Signal Processing, IEEE Transactions on*, vol. 37, no. 12, pp. 1982–1985, 1989.

[40] B. Baas, “A generalized cached-FFT algorithm,” in *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP ’05). IEEE International Conference on*, vol. 5, 2005, pp. v/89–v/92 Vol. 5.

[41] K. K. Parhi, *VLSI Digital Signal Processing Systems - Design and Implementation.* Wiley Inter-Science, 1999.

[42] S. Oraintara, Y. Chen, and T. Nguyen, “Integer fast Fourier transform,” *Signal Processing, IEEE Transactions on*, vol. 50, no. 3, pp. 607–618, 2002.

[43] I. Daubenchies and W. Sweldens, “Factorizing wavelet transforms into lifting steps,” Bell Labs., Lucent Technol., Red Bank, NJ, Tech. Rep., 1996.

[44] C.-H. Lin and A.-Y. Wu, “Mixed-scaling-rotation CORDIC (MSR-CORDIC) algorithm and architecture for high-performance vector rotational DSP applications,” *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, no. 11, pp. 2385–2396, 2005.

[45] C. J. Weinstein, “Quantization effects in digital filters,” MIT Lincoln LAB, Technical report AD0706862, November 1969.

[46] M. Richards, “On hardware implementation of the split-radix FFT,” *Acoustics, Speech, and Signal Processing, IEEE Transactions on*, vol. 36, no. 10, pp. 1575–1581, 1988.

[47] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing.* Prentice-Hall, 1975.

[48] M. Potkonjak, M. Srivastava, and A. Chandrakasan, “Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 15, no. 2, pp. 151–165, 1996.

[49] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford University Press, 2000.

[50] C.-Y. Pai, A. Al-Khalili, and W. Lynch, “Low-power constant-coefficient multiplier generator,” in *ASIC/SOC Conference, 2001. Proceedings. 14th Annual IEEE International*, 2001, pp. 185–189.

[51] J. Garcia, J. Michell, and A. Buron, “VLSI configurable delay commutator for a pipeline split radix FFT architecture,” *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on]*, vol. 47, no. 11, pp. 3098–3107, 1999.

[52] G. Zhang and F. Chen, “Parallel FFT with CORDIC for ultra wide band,” in *Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on*, vol. 2, 2004, pp. 1173–1177 Vol.2.

[53] Y.-W. Lin, H.-Y. Liu, and C.-Y. Lee, “A 1-GS/s FFT/IFFT processor for UWB applications,” *Solid-State Circuits, IEEE Journal of*, vol. 40, no. 8, pp. 1726–1735, 2005.

[54] M. Clark and M. Mulligan, “UWB Fixed-Point Model (Multiband OFDM),” March 2004. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=4577&objectType=file.

[55] Y. Zhang and J. Zhao, “Performance simulation of fixed-point for MB-OFDM UWB system,” in *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on*, vol. 1, 2005, pp. 292–295.

[56] E. H. Wold and A. M. Despain, “Pipeline and parallel pipeline FFT processors for VLSI implementation,” *Computer, IEEE Transactions on*, vol. C-33, pp. 414–424, may 1984.

[57] A. M. Despain, “Fourier transform computer using CORDIC iterations,” *Computer, IEEE Transactions on*, vol. C-23, pp. 993–1001, oct 1974.

[58] T. 3rd Generation Partnership Project (3GPP), "Long term evolution of the 3GPP radio technology," 2004. [Online]. Available: http://www.3gpp.org/Highlights/LTE/LTE.htm.

[59] A. Ghosh, D. Wolter, J. Andrews, and R. Chen, "Broadband wireless access with WiMax/802.16: current performance benchmarks and future potential," *Communications Magazine, IEEE*, vol. 43, no. 2, pp. 129–136, 2005.

[60] F. Behmann, "Impact of wireless (wi-fi, wimax) on 3g and next generation; an initial assessment," in *Electro Information Technology, 2005 IEEE International Conference on*, 2005, pp. 1–6.

[61] Y.-W. Lin and C.-Y. Lee, "A new dynamic scaling FFT processor," in *Circuits and Systems, 2004. Proceedings. The 2004 IEEE Asia-Pacific Conference on*, vol. 1, 2004, pp. 449–452 vol.1.

[62] K. Zhong, H. He, and G. Zhu, "An ultra high-speed FFT processor," in *Signals, Circuits and Systems, 2003. SCS 2003. International Symposium on*, vol. 1, 2003, pp. 37–40 vol.1.